经典计算机
科学著作
最新修订版

计算机程序设计艺术
第3卷 排序与查找

（第2版）

国防工业出版社
National Defence Industry Press
http://www.ndip.com.cn

Addison Wesley

经典计算机科学著作最新修订版

计算机程序设计艺术
第3卷 排序与查找
（第2版）

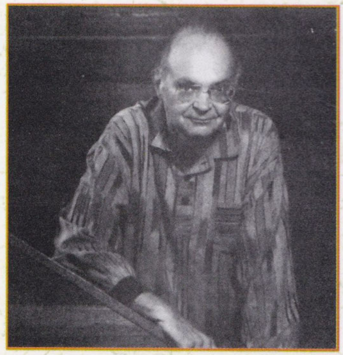
The Art of Computer Programming

苏运霖 译

［美］DONALD E. KNUTH 著

Addison Wesley

国防工业出版社
National Defence Industry Press
http://www.ndip.com.cn



Donald E. Knuth（唐纳德·E·克努特，中文名高德纳）是算法和程序设计技术的先驱者，是计算机排版系统 $\TeX$ 和 METAFONT 的发明者，他因这些成就和大量创造性的影响深远的著作（19部书和160篇论文）而誉满全球。作为斯坦福大学计算机程序设计艺术的荣誉退休教授，他当前正全神贯注于完成其关于计算机科学的史诗性的七卷集。这一伟大工程在1962年他还是加利福尼亚理工学院的研究生时就开始了。Knuth 教授获得了许多奖项和荣誉，包括美国计算机协会图灵奖（ACM Turing Award），美国前总统卡特授予的科学金奖(Medal of Science)，美国数学学会斯蒂尔奖（AMS Steele Prize），以及由于发明先进技术而于1996年11月荣获的极受尊重的京都奖(Kyoto Prize)。现与其妻 Jill 生活于斯坦福校园内。

欲了解这位杰出科学家和作家的更多信息,请访问

www.aw.com/cseng/authors/knuth

欲了解本书和其它各卷的更多信息，请访问

www-cs-faculty.stanford.edu/~knuth

欲了解中译本的更多信息，请访问

www.ndip.com.cn/computer/taocp

经典计算机科学著作最新修订版

计算机程序设计艺术

第3卷 排序与查找

（第2版）

The Art of Computer Programming

苏运霖 译

［美］DONALD E. KNUTH 著

Addison Wesley

国防工业出版社
National Defence Industry Press
http://www.ndip.com.cn

经典计算机科学著作最新修订版

计算机程序设计艺术

第3卷

排序与查找

（第2版）

［美］ Donald E. Knuth 著
苏运霖 译

国防工业出版社
·北京·

著作权合同登记号　图字:军-2001-020 号

图书在版编目(CIP)数据

计算机程序设计艺术．第 3 卷，排序与查找(第 2 版)/(美)克努特(Knuth,D.E.)著;苏运霖译．—北京:国防工业出版社,2002(2003.4)

书名原文:The Art of Computer Programming

ISBN 7-118-02812-6

Ⅰ.计...　Ⅱ.①克...②苏...　Ⅲ.程序设计 -

Ⅳ.TP311

中国版本图书馆 CIP 数据核字（2002）第 004601 号

Original English language title: The Art of Computer Programming, Vol. 2, Seminumerical Algorithms, 3rd Edition by Donald E. Knuth

互联网网页 http://www-cs-faculty. stanford. edu/~knuth/taocp. html 包含本书及相关著作的最新信息。

国防工业出版社出版发行
(北京市海淀区紫竹院南路 23 号)
(邮政编码　100044)
北京奥隆印刷厂印刷
新华书店经售
*
开本 787×960　1/16　印张 50½　插页 1　1041 千字
2002 年 9 月第 1 版　　2003 年 4 月北京第 2 次印刷
印数:4001—7000 册　　定价:98.00 元

前　言

Cookery is become an art,
a noble science;
cooks are gentlemen.
烹调法成了一种艺术,
一门高深的科学;
厨师是有教养的人。
— TITUS LIVIUS, *Ab Urbe Condita* XXXIX. vi
(Robert Burton, *Anatomy of Melancholy* 1.2.2.2)

本书形成了第 1 卷第 2 章关于信息结构内容的自然续篇,因为它为第 1 卷的基本结构化思想增加了线性有序数据的概念。

本书书名“排序与查找”可能给人以印象,仿佛它仅仅是为那些关心通用排序程序编制或信息检索应用的系统程序员编写的。然而,事实上排序与查找的领域还提供了一个理想的园地,以讨论范围广泛而且很重要的一般性问题:

- 如何发现好的算法?
- 如何改进给定的算法或程序?
- 如何从数学上分析算法的效率?
- 对于相同的任务,人们如何在不同的算法之间进行合理的选择?
- 在什么意义之下,可以证明某个算法是“最好”的?
- 计算理论同实际考虑如何相互影响?
- 像磁带、磁鼓或磁盘这样的外存,如何能有效地用于大型数据库?

其实,我确信,程序设计的每一个重要方面,实际上都离不开排序或查找!

本卷由全套书的第 5 章和第 6 章组成。第 5 章讨论的是排序;这是一个很大的课题,已经把它主要划分成两大部分,即内部排序和外部排序。还有补充的几节,提出了关于排列(5.1 节)和最优排序算法(5.3 节)的辅助理论。第 6 章讨论了在表或文件中查找特定项目的问题;这又细分成顺序查找、通过键码(key)比较查找、按数字性质查找、“散列”查找等几种方法,而后考虑了比较困难的辅键码(secondary key)查找问题。在这两章之间存在着令人惊奇的相互影响以及强烈的类似。除了在第 2 章中所讨论的信息结构以外,这里还讨论了信息结构的两种重要的变形,即优先队列(5.2.3 小节)和表示成平衡树的线性表(6.2.3 小节)。

和第 1 卷及第 2 卷一样,本书包括了大量在其它出版物上从未出现过的许多内

容。许多人善意地写信或口头向我表达了他们的看法，我希望，当我以我自己的文字来表现这些内容的时候，我没有很糟糕地曲解它们。

我未曾有时间系统地查找专利文献；确实，我蔑视当前寻求算法专利的倾向(参见 5.4.5 小节)。如果某人把在本书中未曾引用的有关专利的一个副本寄给我，那我将在未来的版本中恭敬地引用它。然而我要鼓励人们继续发扬悠久的数学传统，把最新发现的算法投进公众领域。与其防范他人从某个人对计算机科学的贡献中获益，不如寻求其它更好的谋生手段。因为它与防范自己的计算机成果被他人利用相比会给人们带来更好的生活。

在我从教学工作上退休下来之前，我曾把这本书用做大学三年级至研究生层次数据结构第二门课的教材，并省略掉了大部分数学内容。我也曾使用本书的数学部分作为研究生层次算法分析课程的基础，并特别强调了 5.1，5.2.2，6.3 和 6.4 诸节。关于具体的计算复杂性方面的研究生课程也可以以 5.3 节和 5.4.4 小节，以及第 2 卷的 4.3.3，4.6.3 和 4.6.4 诸小节为基础。

除了偶尔讨论第 1 卷中说明的 MIX 计算机外，本书的绝大部分内容都是独立、自成体系的。附录 B 包含有对于所使用的数学记号的概述，其中的一些记号同传统的数学书中找到的记号有些差别。

第 2 版前言

这个新版本是同第 1 卷及第 2 卷的第 3 版相匹配的。那两卷的出版用上了 TeX 和 METAFONT 系统，使我得以庆祝这两个专门为本套书开发的出版系统的完成。

本书转换成电子格式使我有机会重新检查正文的每一个字和每一个标点符号。在添加或许某些更成熟的记述的同时，我试图保留原来文字的朝气。已经增加了数十个新的习题，对数十个老习题也给出了新的改进了的答案。变动随处可见，但最主要的是在 5.1.4 小节(关于排列和图表)，5.3 节(关于最优排序)，5.4.9 小节(关于磁盘排序)，6.2.2 小节(关于熵)，6.4 节(关于通用散列法)以及 6.5 节(关于多维树和检索结构)。

 然而，《计算机程序设计艺术》的写作仍然是进行中的工作。关于排序与查找的研究仍然以非凡的速度发展着。因此本书的某些部分加上了“正在施工”的图标以对该部分内容还不是最新表示歉意。例如，如果我今天仍向本科生讲授数据结构，我肯定将花些时间讨论像树积(treap)这样的随机化结构；但在本书中，我只能引用有关这个课题的主要文章，告诉大家将来的 6.2.5 小节的写作计划。我的文件夹里已充满了许多重要的素材，我打算从现在起用大约 17 年时间，把这些素材包含在

第 3 卷最后的辉煌的第 3 版中！但是在此之前我必须首先完成第 4 卷和第 5 卷，而且除非绝对必要，否则我不想使它们的问世再有任何拖延了。

我对于过去 35 年来帮助我搜集和改进本书内容的数百位人们表示衷心的感谢。准备这一新版本的大部分艰苦的工作是由 Phyllis Winkler(他把第 1 版的文本转换成 TEX 的形式)和 Silvio Levy(他编辑文字并绘制了数十张插图)，以及 Jeffrey Oldham(他把原来 250 张以上的插图转换成 METAPOST 格式)完成的。Addison-Wesley 生产部的同仁们也一如既往地提供了极大的帮助。

我已经改正了细心的读者们在第 1 版中发现的，以及天哪，竟无人发觉的那些错误，而且，我已竭尽全力试图避免在新的内容中再引进新的错误。然而，我设想，某些欠缺之处仍在所难免，我想尽早地改正它们。因此我将很乐意地支付 2.56 美元给每一个技术、印刷或历史错误的头一个发现者*。在版权页上所引的网页中包含了已向我报告过的所有错误的更正。

Stanford, California　　D. E. K.
1998 年 2 月

There are certain common Privileges of a Writer,
the Benefit whereof, I hope, there will be no Reason to doubt;
Particularly, that where I am not understood, it shall be concluded,
that something very useful and profound is coucht underneath.
一位作者享有某些公认的特权，
我认为，这件事的好处没有理由加以怀疑；
特别是，在我还不被人们理解的地方，
背后蕴涵着某种非常有用和意义深远的东西。
—JONATHAN SWIFT, *Tale of a Tub*, Preface(1704)

* 中文版已按 2003 年 1 月 25 日的最新勘误表更正了原书的错误。网页 http://www.ndip.com.cn/computer/taocp 上也将列出对中文版的勘误表及从作者处获得的最新信息。——本书责任编辑

关于习题的说明

本套书的习题既可用于自学，也可用于课堂练习。无论是谁，如果想纯粹地通过阅读，而不将所阅读的信息应用到特定问题上，并由此牵引思考先前阅读的内容，就想学到一门学问，纵然可能，那也是很困难的。其次，对于我们自己的发现，我们总是领会得最透。因此，习题形成了这一套书的一个重要部分；我着意使这些习题含有丰富的信息，而且也尽量选择既有趣又有启发性的习题。

在许多书里，容易的习题被随机地混杂于极端困难的问题当中。有时这是不合适的，因为读者预先想要知道一道习题花多少时间——否则他们可能就越过这些习题了事。这一情况的典型例子是由 Richard Bellman 所著的*Dynamic Programming* 一书。这是一本重要的先驱性的论著，其中在某些章的末尾，在“习题和研究题”的标题下，把一组问题收集在一起，而且一些极其平凡的问题出现在一些深入的、还未被解决的问题当中。据说，有人曾问过 Bellman 博士，怎样区分习题和研究题。他回答说：“如果你能解决它，那它就是一道习题，否则它就是一个研究题。”

但在这一类书里把研究问题和很容易的习题都包括进来确有其道理；因此，为使读者免于陷入确定哪是习题哪是研究题的困境，我给习题注明了等级分，以指出困难的程度。这些分数大体具有下列意义：

分数	解释
00	一个极其容易的习题。如果你已理解正文的内容，就可能立即做出回答。这样一道题几乎总是可以“眉头一皱”就把它做出。
10	一个简单的问题，它要求你去思考刚刚学过的内容，但绝不意味着是困难的。你应当有能力在顶多一分钟之内就把它做出。在获得解答的过程中可能要用到笔和纸。
20	一个普通的问题。它检查你对正文内容的基本理解，但你可能需要 15 或 20 分钟才能完整地回答它。
30	一个中等难度和/或复杂的问题。这个题目可能需要两个小时以上的工作才能令人满意地解决。或者甚至更长时间，如果电视机在开着的话。
40	确实是一个十分困难或冗长的问题。在学校里，它将适合于作为一个学期的课程设计。一个学生应当有能力在相当长的时间里来解决它，但这个解不会是平凡的。
50	就作者在编写本书时所知，这是一个还未被令人满意地解决的问题，尽管已经有很多人做了尝试。如果你已经找到了这样一个问题的答

案,你应当把它写出来发表。其次,本书的作者将乐意尽快地听到关于解答的消息(当然,假定它是正确的)。

通过在这个“对数”尺上的内插,其它分数的意义也就清楚了。例如 17 分将表示比普通的题更为简单的一道习题。一个随后被某个读者解决了的 50 分的题在本书后来的版本中将以 45 的分数出现,而且会在互联网上的勘误表中刊出(参见版权页)。

分数除以 5 的余数表示所要求的详细工作量。因此分数是 24 的习题可能要比分数是 25 的题花费更长的时间来求解,但后者将要求更多的创造性。

作者已经认真地试图指定精确的分数,但是提出问题的人要想确切地知道对于求解的另一个人,它的难度如何,肯定是困难的;而且每个人都会有较其他人更为适应的问题类型。只能希望这些分数较好地反映了习题的困难程度,希望读者把它们当做是一般的导引,而不应作为绝对的指标。

本书是为具有不同程度的数学训练和素养的读者写的;因此有些习题是特意为有更多的数学基础的读者提供的。如果一道题的出题动机或涉及的数学概念超越了主要兴趣仅仅是算法编程本身的读者应掌握的程度,则在分数前边加上 M。如果一道习题的答案必须涉及在本书中未予提供的微积分或其它高等数学知识,则对这道题标以“HM”的字母。“HM”标记并不必定意味着困难。

某些习题的前边标有三角符“▶”;这一标志说明是特别有启发性的问题,因而特别予以推荐。当然,并不期望读者/学生来求解所有的习题。所以标出了那些看起来最有价值的部分(这并不意味着贬低其它习题!)。每个读者至少应该尝试去解分数是 10 或者更低的所有问题;箭头可以帮助指出应该对具有较高分数的哪些问题予以优先考虑。

在答案部分中已经列出大多数习题的解答,请明智地使用它们。在你已真正地做出努力亲自解决问题之前,不要去翻答案,除非你确实没有时间来做这一特定的问题。在获得了你自己的解答或者对该题做了郑重的尝试之后,你可能会感到答案是有启发和有帮助的。给出的解答通常十分简短,作者认为你已经认真地通过自己的方法做过求解尝试因而略去了其细节。有时解答所提供的信息比所要求的为少,但通常都是更多的。十分可能,你有比这里给的更好的答案,或者在答案中你发现一个错误。在这样的情况下,作者将乐意知道详细的情况。在本书以后的版本中将在适当地方刊登出这些改进了的答案以及提供这些解的人的姓名。

当做一道习题时,一般地你可以使用前边习题的答案,除非明确地禁止这样做。在对习题打分时,已经考虑到这一点了,因此很可能第 $n+1$ 题的分数比第 n 题还要低,尽管它把第 n 题的结果作为一个特殊情况包括进来。

代码含义			
		00	立即可解的
		10	简单的(一分钟)
		20	一般水平(一刻钟)
▶	特别推荐的	*30*	难度适中
M	面向数学的	*40*	学期设计
HM	要求“高等数学”的	*50*	研究题

习 题

▶1.[*00*] 分数"*M20*"意味着什么？

2.[*10*] 一本教科书中的习题对读者有什么价值？

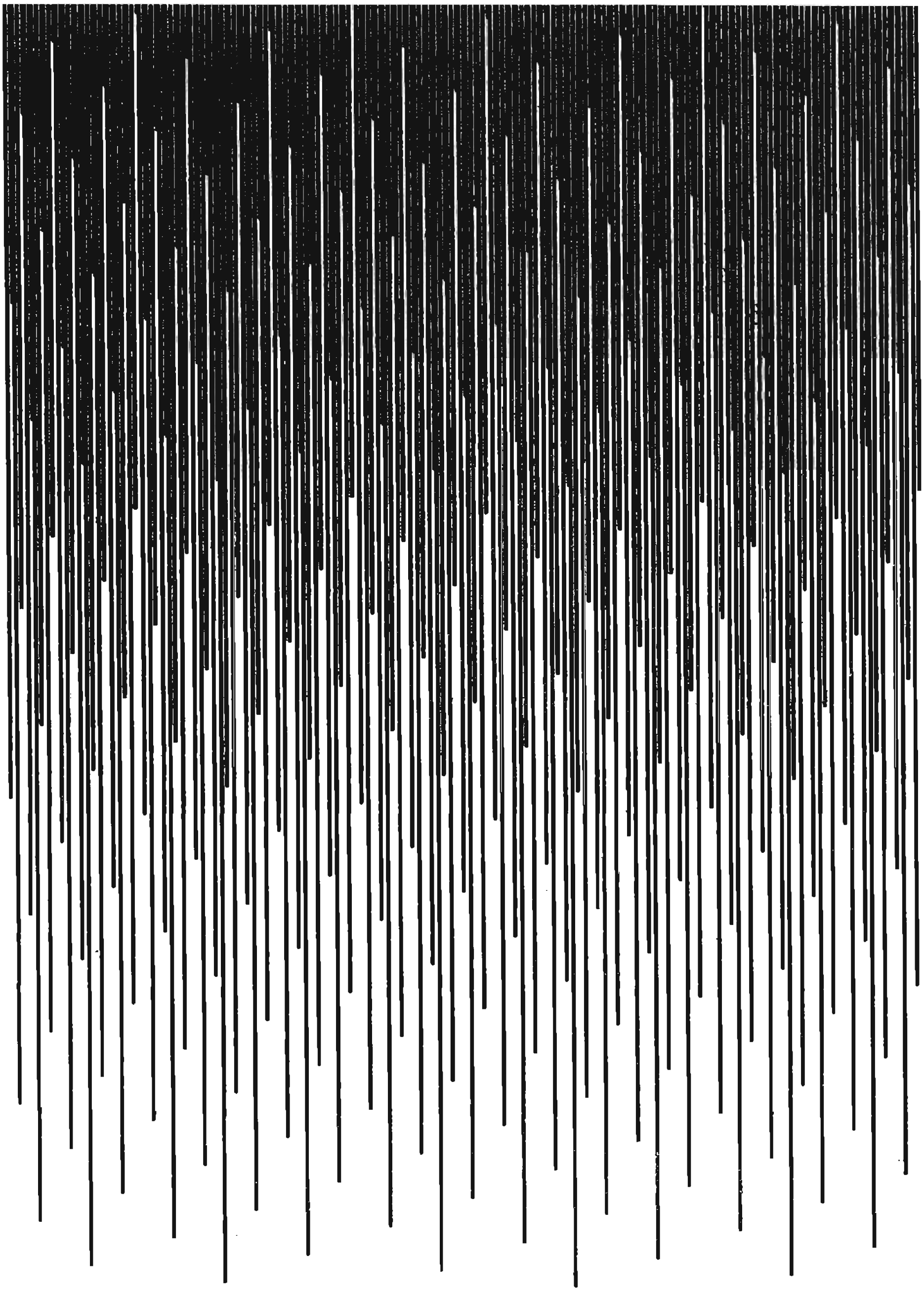
3.[*HM45*] 证明当 n 是一个整数且 $n>2$ 时，方程 $x^n+y^n=z^n$ 无正整数 x,y,z 的解。

Two hours' daily exercise . . . will be enough to keep a hack fit for his work.

每天两小时的训练……将足以令一匹马很好地工作。

—M. H. MAHON, *The Handy Horse Book* (1865)

目　　录



第5章 排　　序

There is nothing more difficult to take in hand,
more perilous to conduct, or more uncertain in its success,
than to take the lead in the introduction of
a new order of things
没有什么比带头建立事物的新秩序更难掌握，
更担风险，更高深莫测的了。
—NICCOLÒ MACHIAVELLI, *The Prince* (1951)

"But you can't look up all those license
numbers in time," Drake objected.
"We don't have to, Paul. We merely arrange a list
and look for duplications."
"但是你不能及时查出全部那些执照号",Drake 争辩说。
"没有必要,Paul。我们只要列一张表,并
把重复的找出来。"
—PERRY MASON, in *The Case of the Angry Mourner* (1951)

"Treesort" Computer—With this new 'computer-approach'
to nature study you can quickly identify over 260
different trees of U. S., Alaska, and Canada,
even palms, desert trees, and other exotics.
To sort, you simply insert the needle.
"树排序"计算机——应用这种新的"计算机方法"
来进行树种研究，你能快速地确认
260 种以上美国、阿拉斯加和加拿大的不同树种，
甚至棕榈树、沙漠里的树，以及其它外来树种。
要想排序，你只须插入这颗针就行了。
—Catalog of Edmund Scientific Company(1964)

在这一章里，我们将研究在程序设计中经常出现的一个课题：以递增或递减的次序重新排列项目。我们可以设想一下，倘若字典中的词不是以字母的顺序排列，

那么使用这样的字典将是何等困难！同样，存在于计算机存储器中的各项的次序，对于处理这些项目的算法的速度及简便性来说，也有着重要的影响。

尽管英语词典里把“sorting”一词定义为按照种类或类型，分开或安排事物的过程，但是计算机程序员习惯于在更为特殊的意义下来使用它，即把事物排成递增或递降的次序。这个过程也许应称做 ordering，而不是 sorting；但任何试图把它称为“ordering”的人很快就会导致混乱，因为这个词已被附加了许多不同的意义。例如，考虑下边的句子：“由于我们只有两台磁带机处于工作状态(working order)，我被要求(was ordered)去以快速订货的形式(in short order)订购(order)更多的磁带机，以便(in order)以快若干数量级(order)的速度对数据进行排序(order)”。在数学术语中，“order”的意义太多了(群的阶(order)，置换的阶(order)，分支点的级数(order)，次序(order)的关系，等等)。所以我们可以得出结论，“order”一词容易导致混乱。

某些人建议用“sequencing”作为排序过程的适当名称，但是这个词通常缺乏恰当的涵义，特别是当出现相等的元素时，而且，它有时同其它术语冲突。实际上，“sorting”本身也是一个被滥用了的词(如：在对那一类(sort)数据进行排序(sorting)以后，我有几分(sort of)不快(out of sorts))，但它已经在计算的用语中被确认并使用了。因此，我们将把“排序”(sorting)一词用于“排成有序”的严格意义之下，而不去作进一步的辩解。

排序的某些最重要的应用是：

a) *解决“一起性”的问题*，即把具有相同标志的所有项目连在一起。假设我们有随机次序的 10 000 个项目，其中许多有相等的值；而且假设我们要重新排列数据，使得具有相等值的所有项出现在连续的位置上。这实际上是该词传统意义下的“sorting”问题；通过在该词的新意义下对文件排序，可以容易地解决这个问题，使得这些值按递增的次序 $v_1 \leqslant v_2 \leqslant \cdots \leqslant v_{10000}$ 排好。在这个过程中可能达到的效率，说明了为什么“sorting”原来的意义已经改变。

b) *匹配在两个或更多个文件中的项*。如果若干个文件已经按次序排好，则通过顺序地扫描一趟它们，就可能从中找出所有匹配的项，而无须返回去查。这是 Perry Mason 在解决一个谋杀案件时用到的原理(见本章开头的引用语)。从开始到末尾顺序地遍历一个信息表，而不是在表中随机地来回跳动，通常可以最快速地处理一个信息表，除非这个表足够小，可以存入到高速随机存取的存储器中。排序使得有可能对大型文件使用顺序存取，从而有可能取代直接寻址。

c) *通过键码值查找信息*。如同我们将在第 6 章中要见到的，排序也有助于查找，因此它也有助于使计算机的输出更适合人们的需要。事实上，按字母顺序排序的一份清单，往往看上去十分可信，即使有关的数字信息计算得不正确也是如此。

尽管排序传统上多用于商业数据处理，但它实际上是程序员应当掌握的运用广泛的一项基本工具。在习题 2.3.2-17 中，我们已经讨论了它在简化代数公式中的用途。以下的习题说明各种各样的典型应用。

展示排序多样性最初的大型软件系统之一，是由 J. Erdwinn 和 D. E. Ferguson，以及他们在 Computer Sciences Corporation(计算机科学公司)的助手们于 1960 年开

发的 LARC Scientific Compiler(拉克科学编译语言)程序。这个用于扩充的 FORTRAN 语言的优化编译程序,大量使用了排序,使得各种编译算法同源程序的有关部分都以适当的顺序出现。第一遍是词法扫描,它把 FORTRAN 源代码分成单个的记号,每个表示一个标志符,或者一个常数,或者一个操作符等等。每个记号被赋以若干顺序号;当对名字和适当的顺序号进行排序时,一个给定标志符的所有使用就联系在一起了。一个用户将用之来确定标志符是否代表一个函数名,一个参数或一个数组变量的"定义项",被给予一个低的顺序号,以使它们在具有一个给定标志符的诸记号当中首先出现;这样,既便于校验一个标志符的使用是否有冲突,又便于相对于 EQUIVALENCE 说明符来分配存储。以这种方式收集在一起的每个标志符的信息现在附加在每个记号上;这样在高速存储器中就不需要保留标志符的"符号表"了。更新过的记号然后按另一个序列号排序,它基本上是把源程序恢复到它原来的顺序,只是该序列是特殊设计的,它把算术表达式变成为更方便的"波兰前缀"形式。排序也用于编译的稍后阶段,以便于进行循环优化,把错误信息合并到清单当中,等等。简言之,编译程序被设计成,使得实际上可以通过顺序地处理存于辅助磁鼓内的文件而完成工作,因为适当的顺序号,是以这样一种方式附加在数据上的,即数据能被排成各种方便的序列。

20 世纪 60 年代的计算机厂家估计,当把他们的所有顾客都考虑在内时,在他们的计算机上,将有超过 25% 的运行时间花在排序上。事实上,有许多计算机装置,仅其中的排序,就用去计算时间的一半以上。由这些统计,可以得出结论:(i)排序有许多重要的应用;(ii)当不应当进行排序时,许多人却进行了;(iii)低效的排序算法正被普遍地使用着。实际情形可能包括所有这 3 种可能性。但无论如何,排序都值得作为一个实际问题而认真地加以研究。

即使排序毫无用处,也仍然有许多理由对它进行研究!业已发现的巧妙算法表明,排序本身就是一个值得剖析的极其有趣的课题。在这方面,还有很多有魅力的悬而未决的问题,当然也有不少已经解决了。

从一个更广泛的方面着眼,我们也会发现,对于一般情况下如何着手解决计算机程序设计的问题,排序算法即是一个有价值的实例分析。在这一章我们将说明数据结构操作的许多重要原理。我们将考察各种排序技术的演化,以期向读者指出,这些思想首先是如何被发现的。通过外推这一实例分析,我们可以学到许多好的策略,来帮助我们就其它计算机问题设计好的算法。

排序技术也为包含于算法分析中的一般思想提供了极好的说明——这些思想通常用来确定算法的性能特征,以便在不同的方法之间进行明智的选择。专长数学的读者,将在这一章中找到用于评价计算机算法的速度和解决复杂的递归关系的,许多有启发的技术。另一方面,由于已对内容做了适当的编排,使得那些并不很擅长数学的读者也能跳过这些计算而不受影响。

在往下进行之前,应该更清楚地来定义我们的问题,并且介绍一些术语。给定有待排序的 N 个项目

$$R_1, R_2, \cdots, R_N$$

我们称这些项目为*记录*，并称 N 个记录的整个集合为一个*文件*。每个记录 R_j 有一个*键码* K_j，它支配着排序的过程。在一个记录中，除这个键码外，还可能有其它的信息；这些额外信息，除非必须作为记录的一部分来一起处理，否则对于排序过程没有影响。

在键码上确定一个次序关系“<”，使得对于任意三个键码值 a,b,c，下列条件成立：

i) $a<b$，$a=b$，$b<a$ 三个可能性中恰有一个可能性成立（这是三分律）。

ii) 如果 $a<b$，而且 $b<c$，则 $a<c$（这是熟知的传递律）。

性质 i) 和 ii) 表征了*线性次序*的数学概念，也称做*全序*。任何满足 i) 和 ii) 的关系“<”都可以按本章中即将介绍的大多数方法进行排序，尽管有些排序技术被设计成只对具有通常次序的数字或字母键码有效。

排序的目标，是确定下标为 $\{1,2,\cdots,N\}$ 的一个排列 $p(1)p(2)\cdots p(N)$，它以非递降的次序来放置所有键码：

$$K_{p(1)} \leqslant K_{p(2)} \leqslant \cdots \leqslant K_{p(N)} \tag{1}$$

如果我们提出进一步的要求，即具有相同键码的记录保留它们原来的相对次序，则称这个排序是*稳定的*。换句话说，稳定排序有下列附加的性质，即

$$p(i) < p(j) \quad 每当\ K_{p(i)} = K_{p(j)}\ 且\ i < j\ 时 \tag{2}$$

在某些情况下我们可能要求在存储器中对记录进行物理上的重新排列，使其键码成为有序的。而在其它一些情况下，可能仅仅需要一张辅助表以某种方式确定这个排列，以便这些记录能根据其键码的次序来加以存取。

本章中有一些排序方法假定了值“∞”或“$-\infty$”，它们分别地定义为大于或小于所有键码：

$$-\infty < K_j < \infty \qquad 对于\ 1 \leqslant j \leqslant N \tag{3}$$

这些极值有时被用做人为的键码，也用做标志（sentinel）指示器。(3) 中排除了相等的情况，如果出现相等，可以对这些算法进行修改，使得它们仍然有效，但通常要损失一些简洁性和有效性。

排序一般可分为*内部排序*和*外部排序*。在内部排序中，记录被整个地保留在计算机的高速随机存取存储器中；当记录比内存一次所能容纳的还多时，则使用外部排序。内部排序在构造和存取数据方面具有更多的灵活性，而外部排序则告诉我们如何应对更为严格的存取约束。

利用一个较好的通用排序算法，对 N 个记录进行排序所需要的时间，大约同 $N\log N$ 成比例；我们对数据大约扫描 $\log N$ “趟”。如同在 5.3.1 小节将看到的，如果所有记录的次序随机而且排序是通过键码的两两比较进行的，则这是极小的时间。于是，如果把记录的数目加倍，则将花略多于两倍的时间对它们进行排序，而所有其它方面不变（实际上，当 N 趋于无穷时，如果键码不同的话，则排序所需时间的一个更好表示是 $N(\log N)^2$，因为键码的大小必须至少以 $\log N$ 的速度增长；但对于实际使用来说，N 事实上绝不会趋于无穷）。

另一方面，如果已知关于某个连续的数值分布，诸键码是随机分布的，则我们将看到，平均说来，我们可以在 $O(N)$ 步内实现排序。

习　题——第一组

1.[*M20*] 由三分律和传递律证明，当假定排序稳定时，排列 $p(1)p(2)\cdots p(N)$ 是惟一确定的。

2.[*21*] 假定某文件中的每个记录 R_j 包含两个键码，一个"主键码" K_j 和一个"次键码" k_j，而且每一个键码集合中各定义了一个线性次序 <。则我们可以用通常的方式在键码对 (K,k) 之间定义一个字典序：

$$(K_i,k_i)<(K_j,k_j)\quad \text{如果 } K_i<K_j \quad \text{或} \quad \text{如果 } K_i=K_j \text{ 且 } k_i<k_j$$

Alice 取这个文件，并且首先按主键码对它进行排序，得到 n 组记录，且对于每组记录，有相同的主键码

$$K_{p(1)}=\cdots=K_{p(i_1)}<K_{p(i_1+1)}=\cdots=K_{p(i_2)}<\cdots<K_{p(i_{n-1}+1)}=\cdots=K_{p(i_n)}$$

其中 $i_n=N$。然后，她按各组的次键码对 n 个组 $R_{p(i_{j-1}+1)},\cdots,R_{p(i_j)}$ 中的每一个进行排序。

Bill 取同一个原始文件，首先按次键码进行排序，然后对得到的文件按主键码进行排序。

Chris 取同一个原始文件，按主键码和次键码 (K_j,k_j) 的字典序对它进行一次排序操作。

问每个人是否得到相同的结果？

3.[*M25*] 设 < 是 $K_1,\cdots,K_N$ 上的一个关系，它满足三分律，但不满足传递律。证明，即使没有传递律，也有可能以一种稳定的方式对记录进行排序，且满足条件(1)和(2)；事实上，至少有 3 个满足这些条件的排列方式！

▶4.[*21*] 实际上词典中并未使用严格的字典序，因为大小写字母都必须参与排列。它们要有以下这样一个顺序：

$$\text{a} < \text{A} < \text{aa} < \text{AA} < \text{AAA} < \text{Aachen} < \text{aah} < \cdots < \text{zzz} < \text{ZZZ}$$

说明如何来实现字典序。

▶5.[*M28*] 试对所有的非负整数设计一个二进制编码，使得如果把 n 编码为串 $\rho(n)$，则我们有 $m<n$ 当且仅当 $\rho(m)$ 按字典序小于 $\rho(n)$。而且，对于任何 $m\neq n$，$\rho(m)$ 不应是 $\rho(n)$ 的一个前缀。如果可能，对于所有大的 n，$\rho(n)$ 的长度应当至多为 $\lg n+O(\log\log n)$（如果我们要对文字和数字混合的文本进行排序的话，或者如果我们要把任意大的字母表映射成二进制串的话，则这样一个编码是有用的）。

6.[*15*] B.C.Dull 先生（一个 MIX 程序员）要想知道存在于单元 A 中的数是否大于、小于或等于存在于单元 B 中的数，所以，他写语句"LDA A;SUB B"并测试寄存器 A 是正，是负或是 0。他犯了什么严重错误？他应该如何做？

7.[*17*] 按下列说明写出多精度键码比较的 MIX 子程序：

调用序列：JMP COMPARE

入口条件：rI1 $=n$；对于 $1\leqslant k\leqslant n$，CONTENTS$(\text{A}+k)=a_k$，CONTENTS$(\text{B}+k)=b_k$；

　　假定 $n\geqslant 1$。

出口条件：CI = GREATER，　如果 $(a_n,\cdots,a_1)>(b_n,\cdots,b_1)$；

　　CI = EQUAL，　如果 $(a_n,\cdots,a_1)=(b_n,\cdots,b_1)$；

CI = LESS，　如果$(a_n,\cdots,a_1)<(b_n,\cdots,b_1)$；

rX 和 rI1 可能要受影响。

这里，关系$(a_n,\cdots,a_1)<(b_n,\cdots,b_1)$表示从左到右的字典序；即，对于 $n\geqslant k>j$，有一个下标 j，使得 $a_k=b_k$，而 $a_j<b_j$。

▶8.[*30*] 单元 A 和 B 分别存放两个数 a 和 b。证明，有可能写出一个 MIX 程序，它计算并保存 $\min(a,b)$于单元 C 中，而不使用任何转移操作符。(警告：由于你不可能测试是否出现算术溢出，所以确保无论是 a 值还是 b 值都不会出现溢出，是明智的。)

9.[*M27*] 在 N 个独立地一致分布于 0 和 1 之间的随机变量排为非递减顺序之后，这些变量中第 r 个最小者小于等于 x 的概率是多少？

习　题——第二组

下列每一个习题指出了，在以前计算机还只有较少随机存取内存的时候，计算机程序员所要解决的问题。假设仅有几千字的内存可供利用，通过大约半打的磁带机作为补充(对于排序来说这些磁带机足够了)，试提出解决这个问题的“好”方法。在这样的限制之下工作得很好的算法，证明在现代机器上也是有效的。

10.[*15*] 给你一条含有 100 万个字的磁带。你如何确定该磁带中有多少字不同？

11.[*18*] 你是美国国内税收服务人员，你收到了来自各单位的数百万“信息”表格，报告他们已向人们支付了多少钱；同时，又从人们那里收到了数百万“税收”表格，报告他们已经得到了多少收人。你如何发现没有报告他们所有收人的人？

12.[*M25*](转置一个矩阵) 给你一条含有 100 万个字的磁带，这些字表示按行顺序存储的 1000×1000 矩阵的元素：$a_{1,1}\ a_{1,2}\cdots a_{1,1000}\ a_{2,1}\cdots a_{2,1000}\cdots a_{1000,1000}$。你如何建立一条磁带，其中元素是按列的次序 $a_{1,1}\ a_{2,1}\cdots a_{1000,1}\ a_{1,2}\cdots a_{1000,2}\cdots a_{1000,1000}$存储的(试试看，最多只对数据扫描 10 趟)？

13.[*M26*] 你如何把 N 个字的一个大型文件“洗”成随机排列？

14.[*20*] 你正在用两个计算机系统进行工作，它们对字母数字的排列方式有不同的约定。你如何使一台计算机按另一台计算机使用的次序对字母数字文件进行排序？

15.[*18*] 给你一份出生在美国的附有其出生州名的人员名单，名单上的人很多，你如何计算出每个州出生的人数(假设这个名单中每个人只出现一次)？

16.[*20*] 为了易于改动大型 FORTRAN 程序，你需要设计一个“交叉引用”程序；这样一个程序以 FORTRAN 程序作为输人，并连同一个索引一起把它们打印出来，索引说明程序中每个标志符(即每个名字)的各次使用。你如何来设计这个程序？

▶17.[*33*](图书馆卡片排序) 在实现计算机化的数据库之前，每个图书馆都有一个卡片目录，以便使用者可以找到他们所要的书。但随着图书馆藏书的增加，将目录卡片排序以便于人们使用这项工作变得十分复杂。下面的“按字母顺序的”清单指出了*American Library Association Rules for Filing Catalog Cards* (Chicago，1942)中建议的许多规程：

卡片的正文	注　释
R. Accademia nazionale dei Lincei, Rome	Ignore foreign royalty(except British)
1812; ein historischer Roman.	Achtzehnhundert zwölf

Bibliothèque d'histoire révolutionnaire.	Treat apostrophe as space in French
Bibliothèque des curiosités.	Ignore accents on letters
Brown, Mrs. J. Crosby	Ignore designation of rank
Brown, John	Names with dates follow those without
Brown, John, mathematician	…and the latter are subarranged
Brown, John, of Boston	by descriptive words
Brown, John, 1715—1766	Arrange identical names by birthdate
BROWN, JOHN, 1715—1766	Works "about" follow works "by"
Brown, John, d. 1811	Sometimes birthdate must be estimated
Brown, Dr. John, 1810—1882	Ignore designation of rank
Brown-Williams, Reginald Makepeace	Treat hyphen as space
Brown America.	Book titles follow compound names
Brown & Dallison's Nevada directory.	& in English becomes "and"
Brownjohn, Alan	
Den', Vladimir Éduardovich, 1867-	Ignore apostrophe in names
The den.	Ignore an initial article
Den lieben süssen Mädeln.	…provided it's in nominative case
Dix, Morgan, 1827—1908	Names precede words
1812 ouverture.	Dix-huit cent douze
Le XIXe siècle fran cais.	Dix-neuvième
The 1847 issue of U.S. stamps.	Eighteen forty-seven
1812 overture.	Eighteen twelve
I am a mathematician.	(a book by Norbert Wiener)
IBM journal of research and development.	Initials are like one-letter words
ha-I ha-ehad.	Ignore initial article
Ia; a love story.	Ignore punctuation in titles
International Business Machines Corporation	
al-Khuwārizmī, Muhammad ibn Mūsā, *fl*. 813—846	Ignore initial "al-" in Arabic names
Labour. A magazine for all workers.	Respell it "Labor"
Labor research association	
Labour, *see* Labor	Cross-reference card

McCall's cookbook	Ignore apostrophe in English
McCarthy, John, 1927-	Mc = Mac
Machine-independent computer programming.	Treat hyphen as space
MacMahon, Maj. Percy Alexander, 1854—1929	Ignore designation of rank
Mrs. Dalloway.	"Mrs." = "Mistress"
Mistress of mistresses.	
Royal society of London	Don't ignore British royalty
St. Petersburger Zeitung.	"St." = "Saint", even in German
Saint-Saëns, Camille, 1835—1921	Treat hyphen as space
Ste-Marie, Gaston P	Sainte
Seminumerical algorithms.	(a book by Donald Ervin Knuth)
Uncle Tom's cabin.	(a book by Harriet Beecher Stowe)
U.S. bureau of the census.	"U.S." = "United States"
Vandermonde, Alexandre Théophile, 1735—1796	
Van Valkenburg, Mac Elwyn, 1921-	Ignore space after prefix in surnames
Von Neumann, John, 1903—1957	
The whole art of legerdemain.	Ignore initial article
Who's afraid of Virginia Woolf?	Ignore apostrophe in English
Wijngaarden, Adriaan Van, 1916-	Surname begins with upper case letter

这些规则大多数都有若干例外，而且还有许多其它规则这里未予说明。

如果你的工作就是用计算机对大量图书馆的目录卡片进行排序，并最终维护这样一个庞大的卡片文件，而且你也没有机会去改变卡片归档的这些长期有效的规则，那么你应怎样安排数据，以便于进行排序和合并操作？

18. [*M25*] (E. T. Parker) 欧拉曾经猜测[*Nova Acta Acad. Sci. Petropolitanæ* **13**(1795), 45～63, §3；写于 1778 年]，方程

$$u^6 + v^6 + w^6 + x^6 + y^6 = z^6$$

没有正整数的解 u, v, w, x, y, z。同时，他猜测，对于所有的 $n \geqslant 3$

$$x_1^n + \cdots + x_{n-1}^n = x_n^n$$

没有正整数的解。但是这个猜测已被计算机发现的恒等式 $27^5 + 84^5 + 110^5 + 133^5 = 144^5$ 所否定，参见 L. J. Lander, T. R. Parkin 和 J. L. Selfridge, *Math. Comp.* **21** (1967), 446～459。后来，Noam Elkies 又发现了 $n = 4$ 时无穷多的反例[*Math. Comp.* **51**(1988), 825～835]。你能否想出一个办法，其中的排序将有助于找到当 $n = 6$ 时欧拉猜测的反例？

▶19. [*24*] 给定一个包含大量不同的 30 位二进制字 $x_1, \cdots, x_N$ 的文件，找出其中所有补码对

$\{x_i, x_j\}$的好方法是什么(两个字称为互补的,如果其中一个是 0 的地方另一个是 1,反之亦然;于是,当它们被处理作二进制数时,互补的充要条件是其和为$(11\cdots1)_2$)?

▶20.[*25*] 给定一个含有 1000 个 30 位二进制字 $x_1, \cdots, x_{1000}$的文件,你如何编制一份所有对偶(x_i, x_j)的表,使得除最多两个二进数位外,$x_i = x_j$?

21.[*22*] 你如何去寻找 5 个字母的变位词(anagram),如:CARET,CARTE,CATER,CRATE,REACT,RECTA,TRACE;CRUEL,LUCRE,ULCER;DOWRY,ROWDY,WORDY[除值得注意的集合

APERS,ASPER,PARES,PARSE,PEARS,PRASE,

PRESA,RAPES,REAPS,SPAER,SPARE,SPEAR

外(我们也把法文字 APRÈS 加进去),人们可能希望知道是否有包括 10 个或更多的 5 字母变位词的任何集合]?

22.[*M28*] 给定大量的有向图说明,为了按同构进行分组,可用什么方法(如果在有向图的顶点之间有一一对应,在它们的有向边之间也有一一对应,并保持顶点和有向边之间的邻接关系,则有向图称为是同构的)?

23.[*30*] 在有 4096 个人的某人群中,每个人有大约 100 个相识者。一个文件已经列出所有相识的对偶(关系是对称的,即:如果 x 同 y 相识,则 y 同 x 相识。因此,此文件大约含有 200 000 项)。你怎样设计一个算法,对于给定的 k,列出这群人当中所有 k 个人的团体(一个团体是互相认识的一组人,团体中每个人都与其他人相识)? 假定不存在人数是 25 的团体,因此团体的总数不可能很多。

▶24.[*30*] 具有不同名字的 300 万人紧挨着排在一起,从纽约排到加利福尼亚。给每个参加者发一张纸条,在这张纸条上写下他自己的名字,以及紧挨着他西侧的那个人的名字。在最西端的那个人不知道怎么做,就把纸条扔了;剩下的 2 999 999 张纸条放到一个大篮子中,并送到首都华盛顿的国家档案局。在那里篮子中的内容已完全乱了,之后转移到磁带上。

这时,一个信息学家发现,在磁带上有足够的信息能重新构造原来顺序下人们的名单。一个计算机学家也发现,通过对整个数据带进行不到 1000 次的扫描,即可重新构造此序列,而且此方法只对磁带文件做顺序存取,只使用少量的随机存取存储器。要如何做才可能呢?

[换言之,对于 $1 \leqslant i < N$,给定随机次序下的对偶(x_i, x_{i+1}),其中,x_i 各不相同,若把所有操作限制为适合使用磁带的串行技术,如何得到序列 $x_1 x_2 \cdots x_N$? 这是当没有容易的方法来确定两个给定的键码中哪一个在前时的排序问题;我们已经把这个问题作为习题 2.2.3-25 的一部分。]

25.[*M21*](**离散对数**) 你知道,p 是一个(相当大的)素数,a 是一个模 p 的原根。因此,对于 $1 \leqslant b < p$ 的所有 b,有惟一的 n 使得 $a^n \bmod p = b$,$1 \leqslant n < p$(这个 n 称做关于 a,b 的模 p 的指数)。给定 b,试说明,怎样在少于 $\Omega(n)$步内找出 n[提示:设 $m = \lceil \sqrt{p} \rceil$,试对 $0 \leqslant n_1, n_2 < m$ 求解 $a^{mn_1} \equiv ba^{-n_2}$ (modulo p)]?

*5.1 排列的组合性质

有限集合的一个排列就是把它的元素排成一行的一种排法。在研究排序算法中,排列具有特殊的重要性,因为它们表示了未排序的输入数据。为了研究不同排序方法的有效性,我们要能计算这样的排列数目,即它们引起一个排序过程的某一步执行一定的次数。

当然，在第 1、第 2 和第 3 章中我们已经数次涉及到排列。例如，在 1.2.5 小节中，讨论了构造 n 个对象的 $n!$ 个排列的两个基本理论方法；在 1.3.3 小节，分析了涉及排列的循环结构和乘法性质的某些算法；在 3.3.2 小节，研究了它们的“上运行”和“下运行”。这一节的目的，是研究排列的其它一些性质，并探讨允许出现相等元素的一般情况。在进行这一研究的过程中，我们将大量地学习组合数学。

排列的性质就其本性来说是非常有趣的，因而，不把其内容分散在整个一章中，而在本书中集中系统地研究较为合适。对于不专长数学的读者，以及急于研究排序技术的读者，建议他们直接跳到 5.2 节，因为本节实际上同排序的直接联系较少。

*5.1.1　反序

设 $a_1, a_2, \cdots, a_n$ 是集合 $\{1, 2, \cdots, n\}$ 的一个排列，如果 $i<j$，且 $a_i>a_j$，则对偶 (a_i, a_j) 称为排列的一个*反序*（或称反演）；例如，排列 3 1 4 2 有 3 个反序：(3,1)，(3,2) 和 (4,2)。每个反序是“违反排序的”一对元素，所以没有反序的惟一的排列是已排好序的排列 $1\ 2 \cdots n$。同排序的这种联系，是我们对反序如此感兴趣的主要原因，尽管我们已经使用过这个概念来分析动态存储分配的算法（见习题 2.2.2-9）。

反序的概念是 G. Cramer 于 1750 年在研究他的解线性方程组的著名规则时引进的[*Intr. à l'Analyse des Lignes Courbes algébriques* (Geneva: 1750), 657～659；参见 Thomas Muir, *Theory of Determinants* **1**(1906), 11～14]。实际上，Cramer 以下列方式定义了 $n \times n$ 阶矩阵的行列式：

$$\det\begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nn} \end{pmatrix} = \sum (-1)^{\mathrm{inv}(a_1 a_2 \cdots a_n)} x_{1a_1} x_{2a_2} \cdots x_{na_n}$$

对 $\{1, 2, \cdots, n\}$ 的所有排列 $a_1\, a_2 \cdots a_n$ 求和，其中，$\mathrm{inv}(a_1\, a_2 \cdots a_n)$ 是排列中的反序个数。

令 b_j 为位于 j *左边但大于* j 的元素个数，就得到排列 $a_1\, a_2 \cdots a_n$ 的*反序表* $b_1\, b_2 \cdots b_n$。换言之，b_j 是第二个分量为 j 的反序的个数。例如，排列

$$5\ 9\ 1\ 8\ 2\ 6\ 4\ 7\ 3 \tag{1}$$

有反序表

$$2\ 3\ 6\ 4\ 0\ 2\ 2\ 1\ 0 \tag{2}$$

因为 5 和 9 在 1 的左边；5，9，8 在 2 的左边；等等。这个排列全部共有 20 个反序。由定义，数 b_j 总满足

$$0 \leqslant b_1 \leqslant n-1,\ 0 \leqslant b_2 \leqslant n-2,\ \cdots,\ 0 \leqslant b_{n-1} \leqslant 1,\ b_n = 0 \tag{3}$$

关于反序，也许最重要的事实是简单的发现，即*一张反序表惟一地确定相应的排列*。我们可以通过逐次地确定元素 $n, n-1, \cdots, 1$（按这个次序）的相对位置，从满足(3)的任何反序表 $b_1\, b_2 \cdots b_n$ 返回到产生它的惟一的排列。例如，我们可以构造对应于(2)的排列如下：写下数字 9，则因 $b_8=1$，所以 8 跟着 9。因 $b_7=2$，把 7 放在

8 和 9 之后。因 $b_6=2$,6 跟在已经写下的头 2 个数之后,所以到此有

$$9\ 8\ 6\ 7$$

继续把 5 置于左边,因 $b_5=0$;置 4 在 4 个数之后;置 3 在 6 个数之后(即在最右边);我们就有

$$5\ 9\ 8\ 6\ 4\ 7\ 3$$

类似地插入 2 和 1 就得到(1)。

这个对应是重要的,因为我们经常能把借助于排列叙述的问题,翻译成借助于反序表叙述的一个等价问题,而后一问题可能比较易于解决。例如,考虑最简单的问题:$\{1,2,\cdots,n\}$可能有多少排列? 回答一定是可能的反序表个数,而这是容易枚举的。因为对于 b_1 有 n 种选择,对 b_2 独立地有 $n-1$ 种选择,……,对 b_n 有一种选择,所以全部共有 $n(n-1)\cdots1=n!$ 种选择。由于诸 b 是完全相互独立的,而诸 a 必须是彼此不同的,所以,反序容易计算。

在 1.2.10 小节,我们分析了当从右到左读一个排列时,该排列的局部极大值个数的问题;换言之,我们要计算有多少个元素大于它们的所有后继(例如,(1)中从右到左的极大值是 3,7,8 和 9)。这个数就是使 b_j 有它的极大值(即 $n-j$)的 j 的个数。因为 b_1 将以 $1/n$ 的概率等于 $n-1$,而 b_2 将(独立地)以 $1/(n-1)$ 的概率等于 $n-2$,等等,显然,由反序的考虑,从右到左的极大值的平均个数是

$$\frac{1}{n}+\frac{1}{n-1}+\cdots+1=H_n$$

类似地,也容易导出对应的生成函数。

如果我们交换一个排列的两个相邻元素,则容易看出,反序的总数将增 1 或减 1。图 1 示出了$\{1,2,3,4\}$的 24 个排列,及其因互换相邻元素而各异的排列的连线;沿任何一条线往下使反序的数目增 1。因此,一个排列 π 的反序数是图 1 中从 1 2 3 4 到 π 的向下通路的长度;所有这样的通路必须有相同的长度。

顺便指出,图 1 的这个图形可以看做三维的立体"截八面体",它有 8 个六边形的面和 6 个正方形的面。这是阿基米德(Archimedes)讨论过的均匀多面体之一(见习题 10)。

读者应不至于把一个排列的反序同一个排列的逆相混淆。回想一下我们可以以两行的形式写一个排列

$$\begin{pmatrix}1 & 2 & 3 & \cdots & n\\ a_1 & a_2 & a_3 & \cdots & a_n\end{pmatrix}\tag{4}$$

这个排列的逆 $a'_1a'_2a'_3\cdots a'_n$,是通过交换这两行,而后对各列进行排序,使得新的顶上那行按递增次序排列所得到的排列

$$\begin{pmatrix}a_1 & a_2 & a_3 & \cdots & a_n\\ 1 & 2 & 3 & \cdots & n\end{pmatrix}=\begin{pmatrix}1 & 2 & 3 & \cdots & n\\ a'_1 & a'_2 & a'_3 & \cdots & a'_n\end{pmatrix}\tag{5}$$

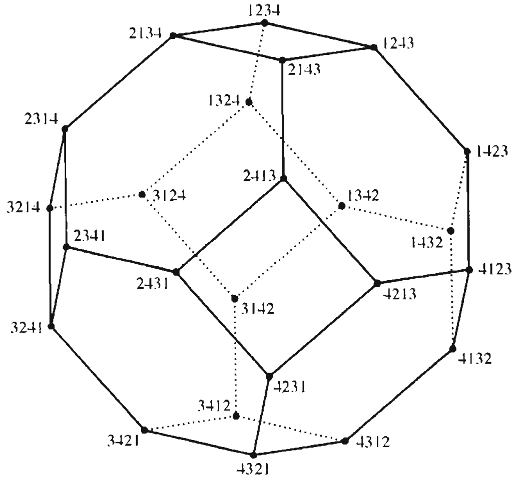


图 1　截八面体,它说明当交换排列的相邻元素时反序的变化

例如,由于

$$\begin{pmatrix}5 & 9 & 1 & 8 & 2 & 6 & 4 & 7 & 3\\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9\end{pmatrix} = \begin{pmatrix}1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9\\ 3 & 5 & 9 & 7 & 1 & 6 & 8 & 4 & 2\end{pmatrix}$$

所以,5 9 1 8 2 6 4 7 3 的逆是 3 5 9 7 1 6 8 4 2。定义逆的另一种方法是说 $a_j' = k$ 当且仅当 $a_k = j$。

排列的逆首先是由 H. A. Rothe[*Sammlung combinatorisch-analytischer Abhandlungen*, C. F. Hindenburg 编,**2** (Leipzig:1800),263～305]定义的。他注意了逆和反序之间一个有趣的关系:*一个排列的逆恰与排列本身有同样多的反序*。Rothe 对这个问题的证明不是最简单的,但它是有教益而且还是相当漂亮的。我们构造一个 $n \times n$ 的棋盘,并且每当 $a_i = j$ 时就在 i 行 j 列处点一个圆点。然后,在那些下边(在同一列)和右边(在同一行)同时都有圆点的所有正方形中打上×。例如,对于 5 9 1 8 2 6 4 7 3 的图为

×	×	×	×	●				
×	×	×	×		×	×	×	●
●								
	×	×	×		×	×	●	
	●							
		×	×		●			
		×	●					
		×				●		
		●						

由于容易看出，b_j 是 j 列中 × 的个数，所以 × 的个数就是反序的个数。现在如果我们把这个图转置一下——即交换行与列——就得到对应原排列的逆的图；因此，在两种情况下，× 的个数（反序的个数）是相同的。Rothe 利用这一事实证明，当对矩阵进行转置时，矩阵的行列式不变。

若干排序算法的分析涉及如下知识：有多少种 n 个元素的排列恰有 k 个反序。让我们用 $I_n(k)$ 表示此数；表 1 列出了这个函数开头的一些值。

表 1 具有 k 个反序的排列

n	$I_n(0)$	$I_n(1)$	$I_n(2)$	$I_n(3)$	$I_n(4)$	$I_n(5)$	$I_n(6)$	$I_n(7)$	$I_n(8)$	$I_n(9)$	$I_n(10)$	$I_n(11)$
1	1	0	0	0	0	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0	0	0	0	0
3	1	2	2	1	0	0	0	0	0	0	0	0
4	1	3	5	6	5	3	1	0	0	0	0	0
5	1	4	9	15	20	22	20	15	9	4	1	0
6	1	5	14	29	49	71	90	101	101	90	71	49

通过考虑反序表 $b_1\, b_2\ \cdots\ b_n$，显然有 $I_n(0)=1$，$I_n(1)=n-1$，而且有对称性质

$$I_n\left(\binom{n}{2}-k\right) = I_n(k) \tag{6}$$

此外，由于每一个 b 可独立于其它诸 b 来选择，故不难看出，生成函数

$$G_n(z) = I_n(0) + I_n(1)z + I_n(2)z^2 + \cdots \tag{7}$$

满足 $G_n(z)=(1+z+\cdots+z^{n-1})G_{n-1}(z)$；因此，它具有由 O. Rodriguez[*J. de Math.* **4** (1839), 236～240]所注意到的比较简单的形式

$$(1+z+\cdots+z^{n-1})\cdots(1+z)(1) = (1-z^n)\cdots(1-z^2)(1-z)/(1-z)^n \tag{8}$$

由这个生成函数，我们可以容易地推广表 1，而且可以验证，该表中锯齿形线下边的数满足（这一关系对锯齿形的上边不成立）

$$I_n(k) = I_n(k-1) + I_{n-1}(k), \quad \text{对于 } k < n \tag{9}$$

更复杂的论证表明（见习题 14），事实上，我们有公式

$$I_n(2) = \binom{n}{2} - 1, \qquad n \geqslant 2$$

$$I_n(3) = \binom{n+1}{3} - \binom{n}{1}, \qquad n \geqslant 3$$

$$I_n(4) = \binom{n+2}{4} - \binom{n+1}{2}, \qquad n \geqslant 4$$

$$I_n(5) = \binom{n+3}{5} - \binom{n+2}{3} + 1, \quad n \geqslant 5$$

一般地说，$I_n(k)$的公式包含有大约 $1.6\sqrt{k}$项：

$$I_n(k) = \binom{n+k-2}{k} - \binom{n+k-3}{k-2} + \binom{n+k-6}{k-5} - \binom{n+k-8}{k-7} - \cdots +$$

$$(-1)^j\left(\binom{n+k-u_j-1}{k-u_j} + \binom{n+k-u_j-j-1}{k-u_j-j}\right) + \cdots, \quad n \geqslant k \tag{10}$$

其中，$u_j = (3j^2 - j)/2$ 就是所谓的“五边形数”。

如果把 $G_n(z)$除以 $n!$，则我们就得到了在 n 个元素的随机排列中，反序个数的概率分布的生成函数，这是乘积

$$g_n(z) = h_1(z)h_2(z)\cdots h_n(z) \tag{11}$$

其中，$h_k(z) = (1 + z + \cdots + z^{k-1})/k$，是小于 k 的一个随机非负整数的一致分布的生成函数。由此得出

$$\text{mean}(g_n) = \text{mean}(h_1) + \text{mean}(h_2) + \cdots + \text{mean}(h_n) =$$

$$0 \quad + \frac{1}{2} \quad + \cdots + \frac{n-1}{2} = \frac{n(n-1)}{4} \tag{12}$$

$$\text{var}(g_n) = \text{var}(h_1) + \text{var}(h_2) + \cdots + \text{var}(h_n) =$$

$$0 \quad + \frac{1}{4} \quad + \cdots + \frac{n^2-1}{12} = \frac{n(2n+5)(n-1)}{72} \tag{13}$$

所以反序的平均数是相当大的，约为$\frac{1}{4}n^2$；标准差也相当大，大约是$\frac{1}{6}n^{3/2}$。

关于反序的分布的一个重大发现是由 P. A. MacMahon 做出的[*Amer, J. Math.*

35 (1913),281～322]。让我们把排列 $a_1\ a_2\ \cdots\ a_n$ 的指数定义为使得 $a_j > a_{j+1}$，$1 \leqslant j < n$ 的所有下标 j 之和。例如，5 9 1 8 2 6 4 7 3 的指数是 2+4+6+8=20。凑巧，在这种情况下，指数和反序个数相同。如果列出{1,2,3,4}的 24 个排列，即

排 列	指数	反序	排 列	指数	反序
1 2 3 4	0	0	3|1 2 4	1	2
1 2 4|3	3	1	3|1 4|2	4	3
1 3|2 4	2	1	3|2|1 4	3	3
1 3 4|2	3	2	3|2 4|1	4	4
1 4|2 3	2	2	3 4|1 2	2	4
1 4|3|2	5	3	3 4|2|1	5	5
2|1 3 4	1	1	4|1 2 3	1	3
2|1 4|3	4	2	4|1 3|2	4	4
2 3|1 4	2	2	4|2|1 3	3	4
2 3 4|1	3	3	4|2 3|1	4	5
2 4|1 3	2	3	4|3|1 2	3	5
2 4|3|1	5	4	4|3|2 1	6	6

则我们看到，*有给定指数 k 的排列的个数，等同于有 k 个反序的排列的个数*。

乍一看，这个事实几乎是显然的，但是进一步仔细研究，就觉得非常不可思议。MacMahon 给出了巧妙的间接证明，如下：设 $\operatorname{ind}(a_1\ a_2\ \cdots\ a_n)$是排列 $a_1\ a_2\ \cdots\ a_n$ 的指数，并令对{1,2,…,n}的所有排列取和的

$$H_n(z) = \sum z^{\operatorname{ind}(a_1 a_2 \cdots a_n)} \tag{14}$$

是对应的生成函数；(14)中的求和是对于{1,2,…,n}中的所有排列进行的。我们希望证明 $H_n(z) = G_n(z)$。为此，我们将定义以非负整数的任意 n 元组$(q_1, q_2, \cdots, q_n)$为一方，和以 n 元组的有序偶

$$((a_1\ a_2\ \cdots\ a_n), (p_1\ p_2\ \cdots\ p_n))$$

为另一方的一一对应，其中 $a_1\ a_2\ \cdots\ a_n$ 是下标{1,2,…,n}的一个排列，且 $p_1 \geqslant p_2 \geqslant \cdots \geqslant p_n \geqslant 0$。这个对应将满足条件

$$q_1 + q_2 + \cdots + q_n = \operatorname{ind}(a_1, a_2, \cdots, a_n) + (p_1 + p_2 + \cdots + p_n) \tag{15}$$

对非负整数的所有 n 元组$(q_1, q_2, \cdots, q_n)$求和的生成函数 $\sum z^{q_1+q_2+\cdots+q_n}$ 为 $Q_n(z) = 1/(1-z)^n$，而对于满足 $p_1 \geqslant p_2 \geqslant \cdots \geqslant p_n \geqslant 0$ 的整数的所有 n 元组$(p_1, p_2, \cdots, p_n)$求和的生成函数$\sum z^{p_1+p_2+\cdots+p_n}$，如习题 15 所示，为

$$P_n(z) = 1/(1-z)(1-z^2)\cdots(1-z^n) \tag{16}$$

依据(15)，我们正要建立的一一对应关系将证明 $Q_n(z) = H_n(z)P_n(z)$，即

$$H_n(z) = Q_n(z)/P_n(z) \tag{17}$$

但由(8)，$Q_n(z)/P_n(z)$为 $G_n(z)$。

我们通过一个简单的排序过程定义所希望的对应关系：以一种稳定的方式，可以把任何 n 元组$(q_1, q_2, \cdots, q_n)$重新安排成非递增的次序 $q_{a_1} \geqslant q_{a_2} \geqslant \cdots \geqslant q_{a_n}$，其中 $a_1\ a_2\ \cdots\ a_n$ 是一个排列，根据这个排列，若 $q_{a_j} = q_{a_{j+1}}$，则意味着 $a_j < a_{j+1}$。我们置 $(p_1, p_2, \cdots, p_n) = (q_{a_1}, q_{a_2}, \cdots, q_{a_n})$，然后，对于 $1 \leqslant j < n$，且对于每一个 j，有 $a_j > a_{j+1}$，将 $p_1, \cdots, p_j$ 中的每一个减 1。这时我们仍然有 $p_1 \geqslant p_2 \geqslant \cdots \geqslant p_n$，因为每当 $a_j > a_{j+1}$时，p_j 都严格地大于 p_{j+1}。得到的对偶$((a_1, a_2, \cdots, a_n), (p_1, p_2, \cdots, p_n))$满足(15)，因为 p 的总的减少等于 $\operatorname{ind}(a_1\ a_2\ \cdots\ a_n)$。例如，如果 $n = 9$，且$(q_1, \cdots, q_9) = (3,1,4,1,5,9,2,6,5)$，则我们可求得 $a_1 \cdots a_9 = 6\ 8\ 5\ 9\ 3\ 1\ 7\ 2\ 4$，$(p_1, \cdots, p_9) = (5,2,2,2,2,2,1,1,1)$。

反之，当给出 $a_1\ a_2\ \cdots\ a_n$ 和$(p_1, p_2, \cdots, p_n)$时，我们能很容易地回到$(q_1, q_2, \cdots, q_n)$去(参见习题 17)。所以，所希望的对应关系已被建立，因此 MacMahon 的指数定理得证。

D. Foata 和 M. P. Schützenberger 在 MacMahon 开创性的著作发表大约 65 年后，发现了对 MacMahon 定理的一个令人惊讶的扩充：*有 k 个反序和指数 l 的 n 个元素的排列数目，和有 l 个反序和指数 k 的排列数目相同*。事实上，Foata 和 Schützenberger 发现了在第一类和第二类排列之间的简单一一对应(参见习题 25)。

习 题

1.[*10*] 排列 2 7 1 8 4 5 9 3 6 的反序表是什么？什么样的排列有反序表 5 0 1 2 1 2 0 0？

2.[*M20*] 在经典的 Josephus 问题中(习题 1.3.2-22)，n 个人开始时被安排在一个圆圈上；然后第 m 个人被枪决，这圆圈收缩，而且重复地消灭每第 m 个人直到全部死光为止。得到的枪决次序是$\{1, 2, \cdots, n\}$的一个排列。例如，当 $n = 8$ 和 $m = 4$ 时，次序是 5 4 6 1 3 8 7 2；对应于这个排列的反序表是 3 6 3 1 0 0 1 0。

当枪决每第 m 个人时，在对于 n 个人的一般的 Josephus 问题中，给出反序表的元素 $b_1\ b_2\ \cdots\ b_n$ 的一个简单递推关系。

3.[*18*] 如果排列 $a_1\ a_2\ \cdots\ a_n$ 对应于反序表 $b_1\ b_2\ \cdots\ b_n$，什么是对应于反序表

$$(n-1-b_1)(n-2-b_2)\cdots(0-b_n)$$

的排列 $\bar{a}_1\ \bar{a}_2\ \cdots\ \bar{a}_n$？

▶4.[*20*] 试设计适合于计算机实现的一个算法，它构造对应于满足(3)的一给定反序表 $b_1\ b_2\ \cdots\ b_n$ 的排列 $a_1\ a_2\ \cdots\ a_n$。[提示：考虑链接存储(linked-memory)技术。]

5.[*35*] 习题 4 的算法在典型的计算机上要求大致与 $n + b_1 + \cdots + b_n$ 成正比的执行时间，因此平均说来，为 $\Theta(n^2)$。考虑是否有算法，其最坏运行时间要比 n^2 阶好得多。

▶6.[*26*] 设计一个算法，它根据$\{1, 2, \cdots, n\}$的一个给定排列 $a_1\ a_2\ \cdots\ a_n$，计算其反序表 $b_1\ b_2$

$\cdots\ b_n$,其在典型计算机上的运行时间基本上同 $n \log n$ 成比例。

7.[*20*] 除了本书定义的特殊反序表 $b_1\ b_2\ \cdots\ b_n$ 外,还可以定义若干其它类型的与{1,2,…,n}的给定排列 $a_1\ a_2\ \cdots\ a_n$ 相对应的反序表;在本题中将考虑在应用中出现的三种其它类型的反序表。

设 c_j 是其第一个分量为 j 的反序的个数,即 j 右边小于 j 的元素个数[对应于(1)我们有表 0 0 0 1 4 2 1 5 7;显然$0 \leqslant c_j < j$]。令 $B_j = b_{a_j}$, $C_j = c_{a_j}$。

证明,对于 $1 \leqslant j \leqslant n$,有 $0 \leqslant B_j < j$ 和 $0 \leqslant C_j \leqslant n-j$;再证明,当给定 $c_1\ c_2\ \cdots\ c_n$ 或 $B_1\ B_2\ \cdots\ B_n$ 或 $C_1\ C_2\ \cdots\ C_n$ 之一时,可惟一地确定排列 $a_1\ a_2\ \cdots\ a_n$。

8.[*M24*] 继续使用习题 7 中的符号,设 $a'_1\ a'_2\ \cdots\ a'_n$ 是排列 $a_1\ a_2\ \cdots\ a_n$ 的逆,并设对应的反序表为 $b'_1\ b'_2\ \cdots\ b'_n$, $c'_1\ c'_2\ \cdots\ c'_n$, $B'_1\ B'_2\ \cdots\ B'_n$ 和 $C'_1\ C'_2\ \cdots\ C'_n$。尽你所能,找出在 $a_j, b_j, c_j, B_j, C_j, a'_j, b'_j, c'_j, B'_j, C'_j$ 之间的许多有趣的关系。

▶9.[*M21*] 在习题 7 的符号下,证明:当且仅当对于 $1 \leqslant j \leqslant n$,有 $b_j = C_j$ 时,$a_1\ a_2\ \cdots\ a_n$ 是一个卷积(即它自己的逆)。

10.[*HM20*] 把图 1 当做一个三维的多面体。如果它的所有边都有单位长度,试问截八面体的直径(顶点 1234 和顶点 4321 之间的距离)是多少?

11.[*M25*] 如果 $\pi = a_1\ a_2\ \cdots\ a_n$ 是{1,2,…,n}的一个排列,设 $E(\pi) = \{(a_i, a_j) \mid i < j, a_i > a_j\}$ 是它的反序集合,并设

$$\bar{E}(\pi) = \{(a_i, a_j) \mid i > j, a_i > a_j\}$$

是那些"非反序"。

a)证明 $E(\pi)$ 和 $\bar{E}(\pi)$ 是传递的(有序对的一个集合 S 称为是传递的,如果每当 (a,b) 和 (b,c) 都在 S 中时,则 (a,c) 在 S 中)。

b)反之,设 E 是 $T = \{(x,y) \mid 1 \leqslant y < x \leqslant n\}$ 的任何传递的子集,它的余集 $\bar{E} = T \setminus E$ 也是传递的。证明存在一个使 $E(\pi) = E$ 的排列 π。

12.[*M28*] 继续使用上题的符号,证明:如果 π_1 和 π_2 是排列,且如果 E 是包含 $E(\pi_1) \cup E(\pi_2)$ 的最小传递集合,则 $\bar{E}$ 是传递的。[因此,如果每当 $E(\pi_1) \subseteq E(\pi_2)$ 时,我们说 π_1 在 π_2"上面",这就定义了排列的一个格;在两个给定的排列"上面",有惟一的"最低"排列。图 1 是当 $n=4$ 时格的图式。]

13.[*M23*] 众所周知,行列式展开中有一半的项有+号,有一半的项有−号。换言之,当 $n \geqslant 2$ 时,具有奇数个反序的排列和具有偶数个反序的排列一样多。试证明,一般情况下,当 $n \geqslant m$ 时,不管整数 t 为何,反序个数模 m 同余于 t 的排列数目为 $n!/m$。

14.[*M24*](F. Franklin(富兰克林)) 把 n 分成 k 个不同部分的分划可用式 $n = p_1 + p_2 + \cdots + p_k$ 表示,其中,$p_1 > p_2 > \cdots > p_k > 0$。例如,把 7 分成不同部分的分划为:7,6+1,5+2,4+3,4+2+1。设 $f_k(n)$ 是把 n 分成 k 个不同部分的分划数;试证明:除非对于某个非负整数 j,n 的形式为 $(3j^2 \pm j)/2$,我们总有 $\sum_k (-1)^2 f_k(n) = 0$;否则,和数是 $(-1)^j$。例如,当 $n=7$ 时,和数是 $-1+3-1=1$,而且 $7 = (3 \times 2^2 + 2)/2$[提示:把一个分划表示成点的一个阵列,对于 $1 \leqslant i \leqslant k$,置 p_i 个点于第 i 行中,试求使得 $p_{j+1} < p_j - 1$ 的最小的 j,并把头 j 行最右边的点圈出。如果 $j < p_k$,则这 j 个点通常可被移走,把它们转 45°,并作为新的第 $k+1$ 行放下。另一方面,如果 $j \geqslant p_k$,则第 k 行的点通常可移走。把它们转 45°,并放置到圈出点的右边(见图 2)。这一过程在大多数情况下,把有奇数个行的分划同有偶数个行的分划配成对,所以在和式中仅需考虑未配对的分划。]

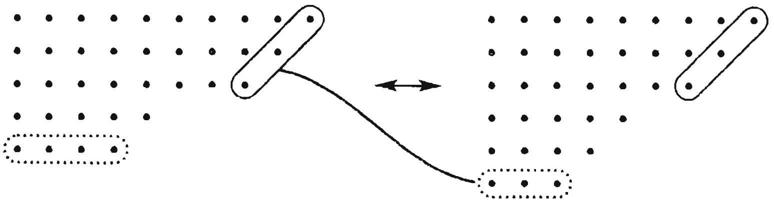


图 2　具有不同部分的分划之间的富兰克林对应

注：作为一个推论，我们得到欧拉公式

$$(1-z)(1-z^2)(1-z^3)\cdots = 1 - z - z^2 + z^5 + z^7 - z^{12} - z^{15} + \cdots = \sum_{-\infty<j<\infty} (-1)^j z^{(3j^2+j)/2}$$

对于通常的分划（它们的各部分不必是不同的）的生成函数为 $\sum p(n)z^n = 1/(1-z)(1-z^2)(1-z^3)\cdots$；因此，我们得到分划数的一个不显然的递推关系式

$$p(n) = p(n-1) + p(n-2) - p(n-5) - p(n-7) + p(n+12) + p(n+15) - \cdots$$

15.[*M23*] 证明(16)是对于至多分成 n 个部分的分划的生成函数。即，证明在 $1/(1-z)(1-z^2)\cdots(1-z^n)$ 中，z^m 的系数是写 $m = p_1 + p_2 + \cdots + p_n$ 的方式个数，其中 $p_1 \geqslant p_2 \geqslant \cdots \geqslant p_n \geqslant 0$。[提示：如在习题 14 中那样画点，证明在 n 元组 $(p_1, p_2, \cdots, p_n)$ $(p_1 \geqslant p_2 \geqslant \cdots \geqslant p_n \geqslant 0)$ 和序列 $(P_1, P_2, P_3 \cdots)$ $(n \geqslant P_1 \geqslant P_2 \geqslant P_3 \geqslant \cdots \geqslant 0)$ 之间有一一对应，并具有性质 $p_1 + p_2 + \cdots + p_n = P_1 + P_2 + P_3 + \cdots$。换言之，分成至多 n 个部分的分划对应于分成不超过 n 的部分的分划。]

16.[*M25*]（L. Euler）借助分划来解释下列恒等式两边，以证明它们：

$$\prod_{k\geqslant 0} \frac{1}{(1-q^k z)} = \frac{1}{(1-z)(1-qz)(1-q^2z)\cdots} = 1 + \frac{z}{1-q} + \frac{z^2}{(1-q)(1-q^2)} + \cdots = \sum_{n\geqslant 0} z^n \Big/ \prod_{1\leqslant k\leqslant n} (1-q^k)$$

$$\prod_{k\geqslant 0} (1+q^k z) = (1+z)(1+qz)(1+q^2z)\cdots = 1 + \frac{z}{1-q} + \frac{z^2 q}{(1-q)(1-q^2)} + \cdots = \sum_{n\geqslant 0} z^n q^{n(n-1)/2} \Big/ \prod_{1\leqslant k\leqslant n} (1-q^k)$$

17.[*20*] 在本节末定义的 MacMahon 对应中，使 $(p_1, p_2, p_3, p_4) = (0,0,0,0)$ 的 24 个四元组 (q_1, q_2, q_3, q_4) 是什么？

18.[*M30*]（T. Hibbard, *CACM* **6** (1963), 210）令 $n > 0$，并设 2^n 个 n 位二进整数 $X_0, \cdots, X_{2^n-1}$ 的序列已经随机地生成，其中每个数的每一个二进位独立地以概率 p 等于 1。考虑序列 $X_0 \oplus 0, X_1 \oplus 1, \cdots, X_{2^n-1} \oplus (2^n - 1)$，其中，$\oplus$ 为对二进表示取“异或”运算。于是，如果 $p=0$，则该序列为 $0, 1, \cdots, 2^n-1$，而且如果 $p=1$，则该序列为 $2^n-1, \cdots, 1, 0$；当 $p=\frac{1}{2}$ 时，这个序列的每个元素是 0 和 2^n-1 之间的一个随机整数。对于一般的 p，这是生成有偏倚的反序个数的随机整数序列的一个有用的方法，尽管整体上序列的元素分布在下述意义上是一致的，即每个 n 位二进整数

有相同的分布。作为概率 p 的一个函数,在这样一个序列中反序的平均个数是多少?

19.[*M28*](C. Meyer) 当 m 与 n 互素时,我们知道,序列 $(m \bmod n)(2m \bmod n)\cdots((n-1)m \bmod n)$ 是 $\{1,2,\cdots,n-1\}$ 的一个排列。证明这个排列的反序的个数可以用戴德金和数来表达(参见 3.3.3 小节)。

20.[*M43*] 下面这一属于雅可比的著名恒等式[*Fundamenta Nova Theoriæ Functionum Ellipticarum* (1829), § 64]是涉及椭圆函数的许多值得注意的关系式的基础:

$$\prod_{k\geqslant 1}(1-u^k v^{k-1})(1-u^{k-1}v^k)(1-u^k v^k) =$$

$$(1-u)(1-v)(1-uv)(1-u^2v)(1-uv^2)(1-u^2v^2)\cdots =$$

$$1-(u+v)+(u^3v+uv^3)-(u^6v^3+u^3v^6)+\cdots =$$

$$\sum_{-\infty<j<+\infty}(-1)^j u^{\binom{j}{2}} v^{\binom{j+1}{2}}$$

例如,如果置 $u=z, v=z^2$,我们可以得到习题 14 的欧拉公式,如果我们置 $z=\sqrt{u/v}, q=\sqrt{uv}$,则得到

$$\prod_{k\geqslant 1}(1-q^{2k-1}z)(1-q^{2k-1}z^{-1})(1-q^{2k}) = \sum_{-\infty<n<\infty}(-1)^n z^n q^{n^2}$$

是否有雅可比恒等式的一个组合证明,该组合证明类似于习题 14 中特殊情况的富兰克林证明?(因此我们要考虑"复数分划"

$$m+n\mathrm{i}=(p_1+q_1\mathrm{i})+(p_2+q_2\mathrm{i})+\cdots+(p_k+q_k\mathrm{i})$$

其中 $p_j+q_j\mathrm{i}$ 是不同的非零复数,p_j 和 q_j 是非负整数且 $|p_j-q_j|\leqslant 1$。雅可比恒等式指出,k 为偶数的这样的表示个数和 k 为奇数的个数一样,除非 m 和 n 是连续的三角形的数。)复数分划有什么值得注意的其它性质?

▶21.[*M25*] (G. D. Knott) 试证明在习题 2.2.1-5 或 2.3.1-6 的意义和习题 7 的符号下,当且仅当对于 $1\leqslant j<n$, $C_j\leqslant C_{j+1}+1$ 时,排列 $a_1 \ \cdots \ a_n$ 可通过一个栈得到。

22.[*M26*] 给定 $\{1,2,\cdots,n\}$ 的一个排列 $a_1 \ \cdots \ a_n$。令 h_j 是使得 $a_i\in\{a_j+1, a_j+2,\cdots,a_{j+1}\}$ 的下标 $i<j$ 的个数(如果 $a_{j+1}<a_j$,这个集合的元素从 n"绕回"到 1。当 $j=n$ 时,我们使用集合 $\{a_n+1,a_n+2,\cdots,n\}$)。例如,排列 5 9 1 8 2 3 4 7 3 导致 $h_1 \ \cdots \ h_9$ = 0 0 1 2 1 4 2 4 6。

a)证明 $a_1 a_2 \ \cdots \ a_n$ 可由数 $h_1 h_2 \ \cdots \ h_n$ 重新构造出来。

b)证明 $h_1+h_2+\cdots+h_n$ 是 $a_1 a_2 \ \cdots \ a_n$ 的下标。

▶23.[*M27*](俄罗斯轮盘赌) 一组较之数论来说更喜欢概率论的应受惩罚的 n 个人,坐成一个圆圈,然后修改 Josephus 方法(习题 2)如下:头一个囚犯握住一支枪并瞄准自己的头部,他以概率 p 死去并离开这个圈子,然后第二个人以相同的方法拿起枪继续进行。这个游戏循环地继续,并且有常数概率 $p>0$,直到每个人都死去为止。

如果 k 这个人是第 j 个死去者,则令 $a_j=k$。试证明死亡的次序 $a_1 a_2 \ \cdots \ a_n$ 以这样一个概率出现,即它仅仅是 n, p 和对偶排列 $(n+1-a_n)\cdots(n+1-a_2)(n+1-a_1)$ 的下标的函数。什么死亡的次序的可能性最小?

24.[*M36*] 给定整数 $t(1)t(2)\cdots t(n)$ 且 $t(j)\geqslant j$,一个排列 $a_1 a_2 \ \cdots \ a_n$ 的推广指数是使得 $a_j>t(a_{j+1})$ 的所有下标之和,加上使得 $i<j$ 和 $t(a_j)\geqslant a_i>a_j$ 的反序的总数。于是对于所有 j,当 $t(j)=j$ 时,推广指数和指数是一样的。但当对于所有 j, $t(j)\geqslant n$ 时,它是反序的数目。试证明其推广指数等于 k 的排列的个数和有 k 个反序的排列的个数相同。[提示:证明,如果我们取

$\{1,\cdots,n-1\}$的任何排列 $a_1\ \cdots\ a_{n-1}$,并且把数 n 插入到所有可能的位置,通过在某个次序之下的数$\{0,1,\cdots,n-1\}$,我们可以增加推广指数。]

▶25.[*M30*](Foata 和 Schützenberger) 如果 $\alpha=a_1\ \cdots\ a_n$ 是一个排列,令 ind(α)是它的指数,inv(α)统计它的反序。

a)定义$\{1,\cdots,n\}$的每个排列 α 到排列 $f(\alpha)$的一一对应关系,其中 $f(\alpha)$有以下两个属性:(i) $\text{ind}(f(\alpha))=\text{inv}(\alpha)$;(ii)对于 $1\leqslant j<n$,数 j 出现在 $f(\alpha)$中 $j+1$ 的左边,当且仅当它出现在 α 中 $j+1$的左边。当 $\alpha=1\ 9\ 8\ 2\ 6\ 3\ 7\ 4\ 5$ 时你的构造指定什么排列到 $f(\alpha)$? 对于什么排列 α,有 $f(\alpha)=1\ 9\ 8\ 2\ 6\ 3\ 7\ 4\ 5$? [提示:如果 $n>1$,写 $\alpha=x_1\alpha_1x_2\alpha_2\cdots x_k\alpha_k a_n$,其中如果 $a_1<a_n$,则 $x_1,\cdots,x_k$全是小于a_n 的元素,否则 $x_1,\cdots,x_n$ 全是大于a_n 的元素;其它的元素出现在(可能是空的)串$\alpha_1,\cdots,\alpha_k$ 中。比较 $h(\alpha)=\alpha_1x_1\alpha_2x_2\cdots\alpha_kx_k$ 到 inv(α)的反序的数目;在这个构造中数 a_n 不出现在$h(\alpha)$中。]

b)使用 f 来定义另一个一一对应关系 g,它有以下两个属性:(i) $\text{ind}(g(\alpha))=\text{inv}(\alpha)$;(ii) $\text{inv}(g(\alpha))=\text{ind}(\alpha)$[提示:考虑逆排列]。

26.[*M25*] 什么是反序的个数和一个随机排列的指数之间的统计关联系数(参见等式 3.3.2-(24))?

27.[*M37*] 证明,除了(15)之外,在 $\text{inv}(a_1\,a_2\ \cdots\ a_n)$和 n 元组$(q_1,q_2,\cdots,q_n)$之间有一个简单的关系。使用这一事实来推广(17)的推导,并得到双变量生成函数的一个代数特征:

$$H_n(w,z)=\sum w^{\text{inv}(a_1a_2\cdots a_n)}z^{\text{ind}(a_1a_2\cdots a_n)}$$

其中这个和是对于所有的 $n!$ 个排列 $a_1\,a_2\ \cdots\ a_n$ 来进行的。

▶28.[*25*](R.W.Floyd,1983) 如果 $a_1\,a_2\ \cdots\ a_n$ 是$\{1,2,\cdots,n\}$的一个排列,它的总位移(Total Displacement)被定义为 $\sum_{j=1}^{n}|a_j-j|$,试借助于反序的数目,求总位移的上限和下限。

29.[*28*] 如果 $\pi=a_1\,a_2\ \cdots\ a_n$ 且$\pi'=a'_1\,a'_2\ \cdots\ a'_n$ 是$\{1,2,\cdots,n\}$的排列,它们的乘积 $\pi\pi'$ 为$a'_{a_1}\,a'_{a_2}\ \cdots\ a'_{a_n}$。设 inv($\pi$)如同在习题 25 中一样,表示反序的数目。试证明 $\text{inv}(\pi\pi')\leqslant\text{inv}(\pi)+\text{inv}(\pi')$,而且等式成立当且仅当在习题 12 的意义下 $\pi\pi'$是在π'"之下"。

*5.1.2 多重集合的排列

至此我们已经讨论了一个元素集合的排列;这仅仅是多重集合排列概念的一种特殊情况(一个多重集合和一个集合一样,只是它可以具有重复的相同元素。在习题 4.6.3-19 中已经讨论了多重集合的某些基本性质)。

例如,考虑多重集合

$$M=\{a,a,a,b,b,c,d,d,d,d\}\tag{1}$$

它包含 3 个 a,2 个 b,1 个 c 和 4 个 d。我们也可以用另外的方式来表示元素多重性,即

$$M=\{3\cdot a,2\cdot b,c,4\cdot d\}\tag{2}$$

M 的一个排列*也是它的元素排成一行的一种排法,例如

* 排列(permutation)有时称为 permatution。——原注

$$c\ a\ b\ d\ d\ a\ b\ d\ a\ d$$

从另一个观点来看,我们又称它为包含3个 a,2个 b,1个 c 和4个 d 的字母串。

M 有多少可能的排列?如果我们把 M 的元素都当做不同的,通过给它们置下标 $a_1,a_2,a_3,b_1,b_2,c_1,d_1,d_2,d_3,d_4$,则将有 $10!=3\,628\,800$ 个排列。但是当去掉这些下标时,它们当中有许多实际上是相同的。事实上,M 的每个排列恰好出现 $3!2!1!4!=288$ 次,因为我们可以由 M 的任何排列开始,以 $3!$ 种方式对 a 置下标,(独立地)以 $2!$ 种方式对 b 置下标,以1种方式对 c 置下标,以及以 $4!$ 种方式对 d 置下标,因此 M 的真正排列个数为

$$\frac{10!}{3!2!1!4!}=12\,600$$

一般说来,通过相同的论证,我们能看到,任何多重集合的排列个数是多项式系数

$$\binom{n}{n_1,n_2,\cdots}=\frac{n!}{n_1!n_2!\cdots} \tag{3}$$

其中 n_1 是一类元素的个数,n_2 是另一类元素的个数,等等;且 $n=n_1+n_2+\cdots$ 是元素的总数。

人们掌握一个集合的排列个数已有1500余年的历史。犹太哲学神秘主义的最早文字著作,Hebrew(希伯莱)的*Book of Creation*(大约公元400年),就给出了头7个阶乘的正确值。在给出此值后,该书说"继续往下就会得到嘴不能说和耳不能听的数了"[*Sefer Yetzirah*,第4章结尾。请见 Solomon Gandz,*Studies in Hebrew Astronomy and Mathematics* (New York: Ktav, 1970), 494～496; Aryeh Kaplan, *Sefer Yetzirah* (York Beach, Maine: Samuel Weiser, 1993)。]这是历史上第一个已知的排列的枚举。第二个出现在印度经典的*Anuyogadvāra-sutra*(约公元500年),规则97,对于既不是递增顺序也不是递降顺序的6个元素的排列的个数,给出公式

$$6\times5\times4\times3\times2\times1-2$$

[参见 G. Chakravarti, *Bull. Calcutta Math. Soc.* **24**(1932), 79～88。*Anuyogadvāra-sutra* 是耆那教规许多书中的一本,该教是在印度兴盛起来的一个宗教派别。]

多重集合的排列的相关公式似乎最初出现于 Bháscara Achárya 的*Lílávatí*(大约公元1150年),270～271中。Bháscara 以稍微简洁的方式指出这个规则,而且仅仅以两个简单的例子{2,2,1,1}和{4,8,5,5,5}来说明它。结果,他的著作的英文翻译并不都能正确地指出该规则,尽管很少有人怀疑 Bháscara 是否真知道他在谈论什么。对于20个数 48555+45855+⋯之和,他曾经给出了有趣的公式

$$\frac{(4+8+5+5+5)\times120\times11111}{5\times6}$$

当只有一个元素被重复时,计算排列数的正确规则是由德国耶稣会学者 Athanasius Kircher 在他关于音乐的长篇论文[*Musurgia Universalis* **2**(Rome, 1650), 5～7]中发现的。他对一个给定的音符集合所能作成的曲调数目很感兴趣,所以,他设想了他所谓的"音乐算术",即从给定音符的给定汇集中作出的变调个数。在他论

文的 18～21 页中，对于若干个 m 和 n 的值，他正确地给出了多重集合 $\{m\cdot C, n\cdot D\}$ 的排列个数，尽管他没有揭示除 $n=1$ 之外他的计算方法。

随后，一般的规则(3)出现于 Jean Prestet 的论文*Eléments de Mathématiques* (Paris:1675)，351～352 中。该书是西方世界写得非常早的解说组合数学的书之一。Prestet 正确地叙述了一般多重集合的规则，但是仅以简单的情况 $\{a,a,b,b,c,c\}$ 来说明它。他特别指出，除以阶乘之和的方法，已行不通了，他认为这种除法是 Kircher 规则的自然的推广。几年之后，John Wallis 在*Discourse of Combinations* (Oxford:1685)的第 2 章(与他的*Treatise of Algebra* 一同出版)，给出了对这一规则的更详细的讨论。

1965 年，Dominique Foata 引进了称为"插入乘积"的一项新颖的思想，使得有可能把关于通常排列的许多已知结果，推广到多重集合排列的一般情况 [见*Publ. Inst. Statistique*, Univ. Paris, **14** (1965), 81～241；也见*Lecture Notes in Math.* **85** (Springer, 1969)。]假定一个多重集合的元素是在某种方式下线性有序的，我们就可以考虑两行的表示法，例如

$$\begin{pmatrix} a\ a\ a\ b\ b\ c\ d\ d\ d\ d \\ c\ a\ b\ d\ d\ a\ b\ d\ a\ d \end{pmatrix} \tag{4}$$

其中，上面一行包含以非减次序排好序的 M 的元素，而下面一行是排列本身。两个多重集合排列 α 和 β 的插入乘积 $\alpha \top \beta$ 定义如下：(a)以两行表示法表达 α 和 β；(b)连接这些两行的表示；(c)把这些列排序，使上行成为非减次序。这个排序应在这样一种意义下是"稳定的"，即：当上面一行的对应元素相等时，下面一行元素自左至右的次序应被保持。例如，$c\ a\ d\ a\ b \top b\ d\ d\ a\ d = c\ a\ b\ d\ d\ a\ b\ d\ a\ d$，因为

$$\begin{pmatrix} a\ a\ b\ c\ d \\ c\ a\ d\ a\ b \end{pmatrix} \top \begin{pmatrix} a\ b\ d\ d\ d \\ b\ d\ d\ a\ d \end{pmatrix} = \begin{pmatrix} a\ a\ a\ b\ b\ c\ d\ d\ d\ d \\ c\ a\ b\ d\ d\ a\ b\ d\ a\ d \end{pmatrix} \tag{5}$$

容易看出，插入乘积运算是可结合的，即

$$(\alpha \top \beta) \top \gamma = \alpha \top (\beta \top \gamma) \tag{6}$$

而且它也满足消去律

$$\begin{aligned} \pi \top \alpha = \pi \top \beta \quad &\text{蕴涵} \quad \alpha = \beta \\ \alpha \top \pi = \beta \top \pi \quad &\text{蕴涵} \quad \alpha = \beta \end{aligned} \tag{7}$$

有一个"恒等元素"

$$\alpha \top \epsilon = \epsilon \top \alpha = \alpha \tag{8}$$

其中 ϵ 是零排列，即空集的"排法"。一般情况下，交换律不成立(见习题 2)，但我们有

$$\alpha \top \beta = \beta \top \alpha \quad \text{如果 } \alpha \text{ 和 } \beta \text{ 没有公共字母} \tag{9}$$

按类似的方式，我们可以把轮换(Cycle)的概念推广到元素重复的情况；令

$$(x_1 \quad x_2 \quad \cdots \quad x_n) \tag{10}$$

表示以一种稳定的方式，根据其顶部的元素把

$$\begin{pmatrix} x_1 & x_2 & \cdots & x_n \\ x_2 & x_3 & \cdots & x_1 \end{pmatrix} \tag{11}$$

的列进行排序，所得到的两行形式的排列。例如我们有

$$(d\ b\ d\ d\ a\ c\ a\ a\ b\ d) = \begin{pmatrix} d\ b\ d\ d\ a\ c\ a\ a\ b\ d \\ b\ d\ d\ a\ c\ a\ a\ b\ d\ d \end{pmatrix} = \begin{pmatrix} a\ a\ a\ b\ b\ c\ d\ d\ d\ d \\ c\ a\ b\ d\ d\ a\ b\ d\ a\ d \end{pmatrix}$$

所以，排列(4)实际上是一个轮换。可以这样描述一个轮换，比方说"d 变为 b 变为 d 变为 d 变为……变为 d 变回来"。注意，这些一般的轮换不具有通常轮换的所有性质；$(x_1\ x_2 \cdots x_n)$不总是和$(x_2 \cdots x_n\ x_1)$一样。

在 1.3.3 小节就注意到，一个集合的每个排列可以惟一地(在不计次序的意义下)表示为不相交轮换的一个乘积，其中排列的"乘积"由一种合成规则定义。容易看出，*不相交轮换的乘积完全等同于它们的插入乘积*；这提示我们，能够推广以前的结果，把一个多重集合的任何排列(在某种意义下)惟一地表示为轮换的插入乘积。事实上，至少有两种自然的方式来做到这一点，每一种都有重要的应用。

等式(5)说明了把 $c\ a\ b\ d\ d\ a\ b\ d\ a\ d$ 分解为较短排列的插入乘积的一条途径；让我们考虑求出一个给定排列 π 的所有分解 $\pi = \alpha \top \beta$ 的一般问题。当研究这个分解因子问题时，考虑一个特殊的排列是有帮助的，例如考虑

$$\pi = \begin{pmatrix} a\ a\ b\ b\ b\ b\ b\ c\ c\ c\ d\ d\ d\ d\ d \\ d\ b\ c\ b\ c\ a\ c\ d\ a\ d\ d\ b\ b\ b\ d \end{pmatrix} \tag{12}$$

如果我们能以形式 $\alpha \top \beta$ 来写这个排列，其中 α 至少含字母 a 一次，则 α 的两行记号的上面一行中，最左边的 a 必须出现于字母 d 之上，所以 α 至少也包含字母 d 的一次出现。如果现在观察 α 上面一行中最左边的 d，则同样看到它必须出现于字母 d 之上。所以，α 必须至少含有两个 d。考察第二个 d，我们看到：α 也必须含有一个 b。在仅仅假定 α 是 π 的一个含有字母 a 的左因子之下，有部分结果

$$\alpha = \begin{pmatrix} a & \cdots & b & \cdots & d\ d & \cdots \\ d & & & & d\ b & \end{pmatrix} \tag{13}$$

以同样方式继续进行，发现(13)上面一行中的 b 必须出现于字母 c 之上，等等。最后，这个过程将再次达到字母 a，并且可以认为这个 a 就是头一个 a，如果我们选定这样做的话。刚才的论证实质上证明了(12)的含有字母 a 的任何左因子 α，有形式 $(d\ d\ b\ c\ d\ b\ b\ c\ a) \top \alpha'$，其中，$\alpha'$ 是某个排列。(把 a 写在这个轮换的最后而不是最前边比较方便。由于仅有一个 a，这是允许的。)类似地，如果假定 α 含有字母 b，则我们就会导出 $\alpha = (c\ d\ d\ b) \top \alpha''$，其中 α'' 是某个排列。

一般说来，这个论证表明，*如果我们有任何因子分解* $\alpha \top \beta = \pi$，*其中* α *含有一个给定的字母* y，*则有形为*

$$(x_1 \cdots\ x_n y), \quad n \geqslant 0, \quad x_1, \cdots, x_n \neq y \tag{14}$$

的惟一轮换，它是 α *的一个左因子*。当给定 π 和 y 时，这个轮换容易确定；它是 π 的含有字母 y 的最短左因子。这项发现的推论之一是下面的定理。

定理 A　*设多重集合 M 的元素对于关系"<"线性有序。M 的每个排列 π 可惟一地表示为下列插入乘积：*

$$\pi = (x_{11}\cdots x_{1n_1}y_1) \top (x_{21}\cdots x_{2n_2}y_2) \top \cdots \top (x_{t_1}\cdots x_{tn_t}y_t),\ t \geqslant 0 \qquad (15)$$

且满足下列两个条件：

$$y_1 \leqslant y_2 \leqslant \cdots \leqslant y_t$$
$$y_i < x_{ij},\quad \text{对于 } 1 \leqslant j \leqslant n_i,\quad 1 \leqslant i \leqslant t \qquad (16)$$

(换句话说,每个轮换中最后的元素小于其它元素,并且最后元素的序列是按非减次序排列的。)

证明　如果 $\pi = \epsilon$,则我们通过令 $t=0$ 而得到这样一个因子分解。否则令 y_1 是被排列的最小元素;而且如上面的例子那样,我们来确定 π 的含 y_1 的最短左因子 $(x_{11}\cdots x_{1n_1}y_1)$。现在,$\pi = (x_{11}\cdots x_{1n_1}y_1)\top\rho$,其中 ρ 为某个排列;通过对长度施行归纳法,可以写出

$$\rho = (x_{21}\cdots x_{2n_2}y_2) \top \cdots \top (x_{t1}\cdots x_{tn_t}y_t),\quad t \geqslant 1$$

它满足(16)的条件,这就证明了这样一个因子分解的存在性。

反之,来证明满足(16)的表示(15)是惟一的。显然,$t=0$,当且仅当 π 是零排列 ϵ。当 $t>0$ 时,(16)意味着 y_1 是被排列的最小元素,而且 $(x_{11}\cdots x_{1n_1}y_1)$ 是含有 y_1 的最短左因子。因此,$(x_{11}\cdots x_{1n_1}y_1)$ 是惟一确定的;由消去律(7)和归纳法可知,这个表示是惟一的。 ▌

例如,若 $a<b<c<d$,则满足给定条件(12)的"典型的"因子分解,是

$$(d\ d\ b\ c\ d\ b\ b\ c\ a) \top (b\ a) \top (c\ d\ b) \top (d) \qquad (17)$$

说明这样一点是重要的,即我们实际上可以把这个表示中的圆括弧和 $\top$ 都去掉,而不致引起二义性！每个轮换恰在剩下的最小元素头一次出现之后结束。所以这个构造把排列

$$\pi' = d\ d\ b\ c\ d\ b\ b\ c\ a\ b\ a\ c\ d\ b\ d$$

同原来的排列

$$\pi = d\ b\ c\ b\ c\ a\ c\ d\ a\ d\ d\ b\ b\ b\ d$$

联系在一起。当 π 的两行表示有形如 $\frac{y}{x}$ 的一个列(其中 $x<y$)时,相关联的排列就有一对应的相邻元素对偶 $\cdots yx\cdots$。因此,我们的例子的排列 π 有形如 $\frac{d}{b}$ 的 3 个列,π' 就有对偶 db 的 3 次出现。一般地说,这个构造建立了下列值得注意的定理。

定理 B　*设 M 是一个多重集合,在 M 的排列之间存在一个一一对应,使得如果 π 对应于 π',则下列条件成立：*

a) *π' 最左边的元素等于 π 最左边的元素；*

b) *对所有满足 $x<y$ 的排列后的元素对偶 (x,y),在 π 的两行表示法中的列 $\frac{y}{x}$*

出现的次数,等于在π'中y直接在x之前的次数。 ▌

当 M 是一个集合时,这实质上就等同于我们在靠近 1.3.3 小节结尾处讨论的"不寻常的对应",只有一些不重要的变化。定理 B 中的更一般的结果对于枚举特殊类型的排列是十分有用的,因为以两行约束为基础来解决一个问题,比起以一个相邻对偶约束为基础来解决等价的问题,通常要更容易些。

P. A. MacMahon 在其卓作 *Combinatory Analysis* **1** (Cambridge Univ. Press, 1915),168～186 中考虑了这种类型的问题。他对 M 仅含两种不同类型的元素——比如说 a 和 b——的特殊情况,给出了定理 B 的一个构造性证明。对于这种情况,他的构造本质上同这里所给出的是一样的,尽管他的表达形式十分不同。对于 3 种不同元素 a,b,c 的情况,MacMahon 给出了定理 B 的一个复杂的非构造性的证明;一般的情况是由 Foata 第一个构造性地证明的[*Comptes Rendlus Acad. Sci.* **258** (Paris,1964),1672～1675]。

作为定理 B 的一个非显然的例子,让我们来求恰巧含有

$$\begin{array}{ll} A & \text{字母 } a \text{ 的 } A \text{ 次出现} \\ B & \text{字母 } b \text{ 的 } B \text{ 次出现} \\ C & \text{字母 } c \text{ 的 } C \text{ 次出现} \\ k & \text{相邻的字母对偶 } ca \text{ 的 } k \text{ 次出现} \\ l & \text{相邻的字母对偶 } cb \text{ 的 } l \text{ 次出现} \\ m & \text{相邻的字母对偶 } ba \text{ 的 } m \text{ 次出现} \end{array} \tag{18}$$

的字母 a,b,c 的串数。本定理告诉我们这等同于形如

$$\left(\begin{array}{c} \overbrace{a\ \cdots\ a}^{A}\quad \overbrace{b\ \cdots\ b}^{B}\quad \overbrace{c\ \cdots\ c}^{C} \\ \underbrace{\underbrace{\underbrace{\sqcup\ \cdots\ \sqcup}_{A-k-m\ a\text{'s}}\quad \underbrace{\sqcup\ \cdots\ \sqcup}_{m\ a\text{'s}}}_{B-l\ b\text{'s}}\quad \underbrace{\underbrace{\sqcup\ \cdots\ \sqcup}_{Rb\text{'s}}}_{l\ b\text{'s}}}_{C\ c\text{'s}} \end{array}\right) \tag{19}$$

的两行数组的个数。这些 a 可以以

$$\binom{A}{A-k-m}\binom{B}{m}\binom{C}{k}$$

种方式放置在第二行中;然后诸 b 可以以

$$\binom{B+k}{B-l}\binom{C-k}{l}$$

种方式放置在剩余的位置上。剩下的空位置必须以诸 c 来填上;因此所求的数目是

$$\binom{A}{A-k-m}\binom{B}{m}\binom{C}{k}\binom{B+k}{B-l}\binom{C-k}{l} \tag{20}$$

让我们回到求出一个给定排列的所有因子分解的问题。是否存在一个"素"排列,即除开它本身和 ϵ 外,没有插入乘积因子的排列? 定理 A 之前的讨论很快地给我们导出了结论,即:*一个排列是素的,当且仅当它是一个无重复元素的轮换*。因为,如果它是这样一个轮换,则我们的论证证明,除开 ϵ 和它本身之外没有左因子。而且如果一个排列含有一个重复的元素 y,则它就有一个其中 y 仅出现一次的非显然的轮换左因子。

一个非素的排列可以被分解成越来越小的块,直到它已被表达成素排列的乘积为止。其次,我们可以证明,如果忽略可交换的因子的次序,则这个因子分解是惟一的。

定理 C　*一个多重集合的每个排列都可以写成一个乘积*

$$\sigma_1 \top \sigma_2 \top \cdots \top \sigma_t, \quad t \geqslant 0 \tag{21}$$

其中,每个 σ_j 是没有重复元素的一个轮换。在下面的意义下,这个表示是惟一的,即同一排列的任何两个这样的表示,可以通过逐次地交换相邻的不相交轮换对偶的办法彼此转换。

"不相交轮换"这一术语意味着没有公共的元素。作为该定理的一个例子,我们可以验证,排列

$$\begin{pmatrix} a\ a\ b\ b\ c\ c\ d \\ b\ a\ a\ c\ d\ b\ c \end{pmatrix}$$

恰有 5 种分解为素因子的方式,即

$$\begin{aligned}(a\,b)\top(a)\top(c\,d)\top(b\,c) = (a\,b)\top(c\,d)\top(a)\top(b\,c) &= \\ (a\,b)\top(c\,d)\top(b\,c)\top(a) &= \\ (c\,d)\top(a\,b)\top(b\,c)\top(a) &= \\ (c\,d)\top(a\,b)\top(a)\top(b\,c)&\end{aligned} \tag{22}$$

证明　我们必须证明,所述的惟一性成立。通过对排列的长度用归纳法,只需证明:如果 ρ 和 σ 是没有重复元素的不相等的轮换,且如果

$$\rho \top \alpha = \sigma \top \beta$$

则 ρ 和 σ 是不相交的,而且对于某个排列 θ,有

$$\alpha = \sigma \top \theta, \beta = \rho \top \theta$$

如果 y 是轮换 ρ 的任何元素,则含有元素 y 的 $\sigma \top \beta$ 的任何左因子,必然含有 ρ 作为一个左因子。所以如果 ρ 和 σ 有一个公共的元素,则 σ 是 ρ 的一个倍数;因此 $\sigma = \rho$(因为它们是素的),这同我们的假定矛盾。故含有 y 且同 σ 没有公共元素的轮换,必然是 β 的一个左因子。通过使用消去律(7),这个证明就完成了。 ▎

作为定理 C 的一个例子,我们考虑由 A 个 a,B 个 b 和 C 个 c 组成的多重集合

$M=\{A\cdot a,B\cdot b,C\cdot c\}$的排列。设$N(A,B,C,m)$是$M$的排列个数，其两行表示法不含有形如$\begin{smallmatrix}a\\a\end{smallmatrix}$，$\begin{smallmatrix}b\\b\end{smallmatrix}$，$\begin{smallmatrix}c\\c\end{smallmatrix}$的列，而恰含有$m$个形如$\begin{smallmatrix}a\\b\end{smallmatrix}$的列。由此得出，形如$\begin{smallmatrix}a\\c\end{smallmatrix}$的列恰有$A-m$个，形如$\begin{smallmatrix}c\\b\end{smallmatrix}$的恰有$B-m$个，形如$\begin{smallmatrix}c\\a\end{smallmatrix}$的有$C-B+m$个，形如$\begin{smallmatrix}b\\c\end{smallmatrix}$的有$C-A+m$个，以及形如$\begin{smallmatrix}b\\a\end{smallmatrix}$的有$A+B-C-m$个。因此：

$$N(A,B,C,m)=\binom{A}{m}\binom{B}{C-A+m}\binom{C}{B-m} \tag{23}$$

定理C告诉我们，可以以另一种方式计算这些排列：由于排除了形如$\begin{smallmatrix}a\\a\end{smallmatrix}$，$\begin{smallmatrix}b\\b\end{smallmatrix}$，$\begin{smallmatrix}c\\c\end{smallmatrix}$的列，这一排列仅有的可能的素因子为

$$(a\ b),(a\ c),(b\ c),(a\ b\ c),(a\ c\ b) \tag{24}$$

这些轮换中的每一对至少有一公共字母，所以分解成素因子是完全惟一的。如果轮换$(a\ b\ c)$在因子分解中出现k次，则以前的假设意味着$(a\ b)$出现$m-k$次，$(b\ c)$出现$C-A+m-k$次，$(a\ c)$出现$C-B+m-k$次，以及$(a\ c\ b)$出现$A+B-C-2m+k$次。因此$N(A,B,C,m)$是这些轮换的排列个数（一个多项式系数），对k求和：

$$N(A,B,C,m)=\sum_k\frac{(C+m-k)!}{(m-k)!(C-A+m-k)!(C-B+m-k)!k!(A+B-C-2m+k)!}=\sum_k\binom{m}{k}\binom{A}{m}\binom{A-m}{C-B+m-k}\binom{C+m-k}{A} \tag{25}$$

同(23)比较，发现下列恒等式一定成立：

$$\sum_k\binom{m}{k}\binom{A-m}{C-B+m-k}\binom{C+m-k}{A}=\binom{B}{C-A+m}\binom{C}{B-m} \tag{26}$$

这是在习题1.2.6-31中遇到的恒等式，即

$$\sum_j\binom{M-R+S}{j}\binom{N+R-S}{N-j}\binom{R+j}{M+N}=\binom{R}{M}\binom{S}{N} \tag{27}$$

其中，$M=A+B-C-m$，$N=C-B+m$，$R=B$，$S=C$，以及$j=C-B+m-k$。

类似地，可以计算$\{A\cdot a,B\cdot b,C\cdot c,D\cdot d\}$的排列个数，使得其中各种类型的列的个数指定如下：

列的类型：	$\begin{smallmatrix}a\\d\end{smallmatrix}$	$\begin{smallmatrix}a\\b\end{smallmatrix}$	$\begin{smallmatrix}b\\a\end{smallmatrix}$	$\begin{smallmatrix}b\\c\end{smallmatrix}$	$\begin{smallmatrix}c\\b\end{smallmatrix}$	$\begin{smallmatrix}c\\d\end{smallmatrix}$	$\begin{smallmatrix}d\\a\end{smallmatrix}$	$\begin{smallmatrix}d\\c\end{smallmatrix}$
次数：	r	$A-r$	q	$B-q$	$B-A+r$	$D-r$	$A-q$	$D-A+q$

(28)

（这里$A+C=B+D$。）于是对于某个s，在这种排列的素因子分解中可能出现的轮

换是(见习题 12)：

$$\begin{array}{lcccccc} \text{轮换：} & (a\ b) & (b\ c) & (c\ d) & (d\ a) & (a\ b\ c\ d) & (d\ c\ b\ a) \\ \text{次数：} & A-r-s & B-q-s & D-r-s & A-q-s & s & q-A+r+s \end{array} \tag{29}$$

在这种情况下，轮换$(a\ b)$和$(c\ d)$可相互交换，而且$(b\ c)$和$(d\ a)$亦然，所以必须计算不同的素因子分解的个数。结果是(见习题 10)，总有一个惟一的因子分解，使得$(c\ d)$之后不会紧跟$(a\ b)$，且$(d\ a)$之后不会紧跟$(b\ c)$。因此，由习题 13 的结果，有恒等式

$$\sum_{s,t}\binom{B}{t}\binom{A-q-s}{A-r-s-t}\binom{B+D-r-s-t}{B-q-s}\times$$
$$\frac{D!}{(D-r-s)!(A-q-s)!s!(q-A+r+s)!}=$$
$$\binom{A}{r}\binom{B+D-A}{D-r}\binom{B}{q}\binom{D}{A-q}$$

从公式两边去掉因子$\binom{D}{A-q}$并稍微简化阶乘，剩下颇显复杂的 5 个参数的恒等式

$$\sum_{s,t}\binom{B}{t}\binom{A-r-t}{s}\binom{B+D-r-s-t}{D+q-r-t}\binom{D-A+q}{D-r-s}\binom{A-q}{r+t-q}=$$
$$\binom{A}{r}\binom{B+D-A}{D-r}\binom{B}{q} \tag{30}$$

利用(27)可实现对 s 求和，而且对 t 的和数也容易计算；所以，在完成所有这些之后，我们未能有幸发现尚不知如何导出的任何恒等式。但是至少我们已经学习了以两种不同的方式计算某些类型的排列，而且这些计算技术对于以后的问题是一个很好的训练。

习　题

1.[*M05*] **真或假**：设 M_1 和 M_2 是多重集合，如果 α 是M_1 的一个排列，β 是M_2 的一个排列，则 $\alpha \top \beta$ 是$M_1 \cup M_2$ 的一个排列。

2.[*10*] 在(5)中计算了 $c\,a\,d\,a\,b$ 与$b\,d\,d\,a\,d$ 的插入乘积；试求当交换因子时得到的插入乘积 $b\,d\,d\,a\,d \top c\,a\,d\,a\,b$。

3.[*M13*] (9)的逆成立否？换言之，如果 α 和β 在插入乘积中可交换，则它们必须没有公共字母吗？

4.[*M11*] 当 $a<b<c<d$ 时，在定理 A 的意义下，(17)中给出了(12)的典型因子分解。试求当 $d<c<b<a$ 时对应的典型因子分解。

5.[*M23*] 定理 B 的条件 b)要求 $x<y$；如果把这个关系减弱为 $x\leqslant y$，则会发生什么情况？

6.[*M15*] 问这样的串有多少：它恰含有 m 个a，n 个b，又无其它字母，而且恰在 k 个a 的前面都有一个b？

7.[*M21*] 满足条件(18)且以字母 a 打头的字母 a,b,c 的串有多少？以字母 b 打头的呢？以字母 c 打头的呢？

▶8.[*20*] 求把(12)分解成两个因子 $\alpha \top \beta$ 的所有因子分解。

9.[*33*] 试写出把一个给定的多重集合的排列，分解成定理 A 和定理 C 所述的形式的计算机程序。

▶10.[*M30*] 真或假：尽管根据定理 C 分解素因子不真正惟一，但我们可以以下列方式确保惟一性："存在素排列集合的线性次序 $\prec$，使得一个多重集合的每个排列有惟一的素因子分解 $\sigma_1 \top \sigma_2 \top \cdots \top \sigma_n$，它满足条件：对于 $1 \leqslant i < n$，当 σ_i 与 σ_{i+1} 可交换时，$\sigma_i \succeq \sigma_{i+1}$。

▶11.[*M26*] 设 $\sigma_1,\sigma_2,\cdots,\sigma_t$ 是没有重复元素的轮换。如果 $i<j$ 且 σ_i 至少同 σ_j 有一个公共字母，则说 $x_i \prec x_j$，这样就定义了 t 个对象 $\{x_1,\cdots,x_t\}$ 上的偏序 $\prec$。证明定理 C 和 2.2.3 小节拓扑排序的思想之间的下列联系：*$\sigma_1 \top \sigma_2 \top \cdots \top \sigma_t$ 的不同素因子分解的数目，是对给定的偏序进行拓扑排序的方式个数*（例如对应于(22)，我们发现有 5 种方式对次序 $x_1 \prec x_2, x_3 \prec x_4, x_1 \prec x_4$ 拓扑排序）。反之，给定 t 个元素的任何偏序，就有一轮换集合

$$\{\sigma_1,\sigma_2,\cdots,\sigma_t\}$$

以所述的方式定义偏序。

12.[*M16*] 证明(29)是(28)的假设的一个推论。

13.[*M21*] 证明不含有相邻的字母对偶 ca 和 db 的多重集合 $\{A\cdot a, B\cdot b, C\cdot c, D\cdot d, E\cdot e, F\cdot f\}$ 的排列个数是

$$\sum_t \binom{D}{A-t}\binom{A+B+E+F}{t}\binom{A+B+C+E+F-t}{B}\binom{C+D+E+F}{C,D,E,F}$$

14.[*M30*] 根据本小节中其它定义的提示，定义一个一般排列 π 的逆 π^- 的方法之一，是把 π 的两行表示的行交换，然后对这些列做稳定排序，以便把上面的行变为非减次序。例如，如果 $a<b<c<d$，则这个定义意味着 $c\,a\,b\,d\,d\,a\,b\,d\,a\,d$ 的逆是 $a\,c\,d\,a\,d\,a\,b\,b\,d\,d$。

试剖析这个求逆操作的性质；例如，它同插入乘积是否有任何简单的关系？我们能计算使 $\pi=\pi^{-1}$ 的排列的数目吗？

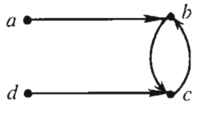
▶15.[*M25*] 证明多重集合

$$\{n_1\cdot x_1, n_2\cdot x_2,\cdots,n_m\cdot x_m\}$$

的排列 $a_1\cdots a_n$，其中 $x_1<x_2<\cdots<x_m$，$n_1+n_2+\cdots+n_m=n$ 是一个轮换，当且仅当具有顶点 $\{x_1,x_2,\cdots,x_m\}$，和从 x_j 到 $a_{n_1+\cdots+n_j}$ 的诸有向边的有向图恰好包含一条有向回路。在后一种情况下，以轮换形式表示排列的方式个数是有向回路的长度。例如，对应

$$\begin{pmatrix} a\,a\,a\,b\,b\,c\,c\,c\,d\,d \\ d\,c\,b\,a\,c\,a\,a\,b\,d\,c \end{pmatrix}$$

的有向图是



而把这个排列表示成一个轮换的两种方式是$(b\,a\,d\,d\,c\,a\,c\,a\,b\,c)$和$(c\,a\,d\,d\,c\,a\,c\,b\,a\,b)$。

16.[*M35*] 在排列一个集合的特殊情况下，我们在上节建立了排列的反序的生成函数，即等式 5.1.1-(8)。试证明，一般情况下，如果排列一个多重集合，则 $\{n_1\cdot x_1, n_2\cdot x_2,\cdots\}$ 的反序的生成函数是"z-多项式系数"

$$\binom{n}{n_1, n_2, \cdots}_z = \frac{n!_z}{n_1!_z n_2!_z \cdots}, \qquad \text{其中 } m!_z = \prod_{k=1}^{m}(1+z+\cdots+z^{k-1})$$

[同(3)和等式 1.2.6-(40)中的 z-二项式系数的定义作比较。]

17.[*M24*] 利用在习题 16 中建立的生成函数,求出在一个给定的多重集合的随机排列中,反序个数的平均值和标准离差。

18.[*M30*](P.A.MacMahon) 上一小节定义了一个排列 $a_1 a_2 \cdots a_n$ 的指数;而且证明了一个集合的指数为 k 的排列个数,等同于有 k 个反序的排列个数。对于一个给定的多重集合的排列,同样的结果成立否?

19.[*HM28*] 定义一个排列 π 的 Möbius 函数 $\mu(\pi)$如下:如果 π 含有重复的元素,则 $\mu(\pi)$为零;否则,如果 π 是k 个素因子的乘积,则 $\mu(\pi)$为$(-1)^k$[同通常的 Möbius 函数的定义,即习题 4.5.2-10 进行比较]。

a)证明如果 $\pi\neq\epsilon$,则对所有为 π 的左因子的排列λ 求和(即对某个 ρ,$\lambda \top \rho=\pi$),我们有

$$\sum \mu(\lambda) = 0$$

b)给定 $x_1<x_2<\cdots<x_m$ 和$\pi=x_{j_1}x_{j_2}\cdots x_{j_n}$,其中,对于 $1\leqslant k\leqslant n$,$1\leqslant j_k\leqslant m$,证明

$$\mu(\pi) = (-1)^n \epsilon(i_1\ i_2\ \cdots\ i_n)$$

其中,$\epsilon(i_1\ i_2\ \cdots\ i_n)=\mathrm{sign}\prod_{1\leqslant j<k\leqslant n}(i_k-i_j)$。

▶20.[*HM33*](D.Foata) 设(a_{ij})是任意实数矩阵。在习题 19b)的记号下,定义 $\nu(\pi)=a_{i_1j_1}\cdots a_{i_nj_n}$,其中 π 的两行记号为

$$\begin{pmatrix} x_{i_1} & x_{i_2} & \cdots & x_{i_n} \\ x_{j_1} & x_{j_2} & \cdots & x_{j_n} \end{pmatrix}$$

这个函数在计算一个多重集合排列的生成函数时是有用的,因为对于多重集合

$$\{n_1\cdot x_1,\cdots,n_m\cdot x_m\}$$

的所有排列 π 求和的 $\sum \nu(\pi)$,将是满足某些限制的排列个数的生成函数。例如,如果对于 $i=j$,取 $a_{ij}=z$,以及对于 $i\neq j$,取 $a_{ij}=1$,则 $\sum \nu(\pi)$是对于“不动点”(上面和下面元素相同的列)个数的生成函数。为了同时研究所有多重集合的 $\sum \nu(\pi)$,我们考虑函数

$$G = \sum \pi\nu(\pi)$$

它对于包含元素 $x_1,\cdots,x_m$ 的多重集合的所有排列集合$\{x_1,\cdots,x_m\}^*$中的 π 求和。

在 G 的这个公式中,我们把 π 处理作诸x 的乘积。例如,当 $m=2$ 时,有

$$\begin{aligned} G = {} & 1 + x_1\nu(x_1) + x_2\nu(x_2) + x_1x_1\nu(x_1x_1) + x_1x_2\nu(x_1x_2) + \\ & x_2x_1\nu(x_2x_1) + x_2x_2\nu(x_2x_2) + \cdots = \\ & 1 + x_1a_{11} + x_2a_{22} + x_1^2a_{11}^2 + x_1x_2a_{11}a_{22} + \\ & x_1x_2a_{21}a_{12} + x_2^2a_{22}^2 + \cdots \end{aligned}$$

于是 G 中 $x_1^{n_1}\cdots x_m^{n_m}$ 的系数就是对于 $\{n_1\cdot x_1,\cdots,n_m\cdot x_m\}$ 的所有排列 π 求和的 $\sum\nu(\pi)$。不难看出，这个系数也是在表达式

$$(a_{11}x_1+\cdots+a_{1m}x_m)^{n_1}(a_{21}x_1+\cdots+a_{2m}x_m)^{n_2}\cdots(a_{m1}x_1+\cdots+a_{mm}x_m)^{n_m}$$

中 $x_1^{n_1}\cdots x_m^{n_m}$ 的系数。

这个习题的目的是要证明 P. A. MacMahon 在他的*Combinatory Analysis* **1** (1915)第 3 节中所谓的“主要定理”，即公式

$$G=1/D$$

其中：

$$D=\det\begin{pmatrix}1-a_{11}x_1 & -a_{12}x_2 & \cdots & -a_{1m}x_m\\ -a_{21}x_1 & 1-a_{22}x_2 & \cdots & -a_{2m}x_m\\ \vdots & \vdots & & \vdots\\ -a_{m1}x_1 & -a_{m2}x_2 & \cdots & 1-a_{mm}x_m\end{pmatrix}$$

例如，如果对于所有的 i 和 j，$a_{ij}=1$，则这个公式给出

$$G=1/(1-(x_1+x_2+\cdots+x_m))$$

而 $x_1^{n_1}\cdots x_m^{n_m}$ 的系数正好像它应该的那样，即 $(n_1+\cdots+n_m)!/n!\cdots n_m!$。

为了证明主要定理，证明

a) $\nu(\pi\top\rho)=\nu(\pi)\nu(\rho)$；

b) 在习题 19 的记号下，$D=\sum\pi\nu(\pi)\nu(\pi)$，该式对

$$\{x_1,\cdots,x_m\}^*$$

中的所有排列 π 进行求和；

c) 因此 $D\cdot G=1$。

21. [*M21*] 给定 $n_1,\cdots,n_m$ 和 $d\geqslant 0$，对于 $1\leqslant j<n=n_1+\cdots+n_m$，多重集合 $\{n_1\cdot 1,\cdots,n_m\cdot m\}$ 中有多少排列 $a_1,a_2,\cdots,a_n$ 满足 $a_{j+1}\geqslant a_j-d$？

22. [*M30*] 设 $P(x_1^{n_1}\cdots x_m^{n_m})$ 表示多重集合 $\{n_1\cdot x_1,\cdots,n_m\cdot x_m\}$ 的所有可能排列的集合，并设 $P(x_0^{n_0}x_1^{n_1}\cdots x_m^{n_m})$ 是 $P(x_0^{n_0}x_1^{n_1}\cdots x_m^{n_m})$ 的子集，其中头 n_0 个元素 $\neq x_0$。

a) 给定满足 $1\leqslant t<m$ 的一个数 t，试求对于某个 $k\geqslant 0$，$P(1^{n_1}\cdots m^{n_m})$ 和分别属于 $P_0(0^k1^{n_1}\cdots t^{n_t})$ 和 $P_0(0^k(t+1)^{n_{t+1}}\cdots m^{n_m})$ 的所有有序的排列偶的集合之间的一一对应。[提示：对于每个 $\pi=a_1\cdots a_n\in P(1_1^{n}\cdots m^{n_m})$，令 $l(\pi)$ 是通过以 0 来代替 $t+1,\cdots,m$ 和删除最后的 $n_{t+1}+\cdots+n_m$ 个位置中的 0 所得到的排列；类似地，令 $r(\pi)$ 是以 0 来代替 $1,\cdots,t$ 并且删除头 $n_1+\cdots+n_t$ 个位置中的所有 0 所得到的排列。]

b) 证明其两行形式有 p_j 个列 ^0_j 和 q_j 个列 $^j_0$ 的 $P(0^{n_0}1^{n_1}\cdots m^{n_m})$ 的排列的个数为

$$\frac{|P(x_1^{p_1}\cdots x_m^{p_m}y_1^{n_1-p_1}\cdots y_m^{n_m-p_m})|\,|P(x_1^{q_1}\cdots x_m^{q_m}y_1^{n_1-q_1}\cdots y_m^{n_m-q_m})|}{|P_0(0^{n_0}1^{n_1}\cdots m^{n_m})|}$$

c) 设 $w_1,\cdots,w_m,z_1,\cdots,z_m$ 是单位圆上的复数，把一个排列 $\pi\in P(1^{n_1}\cdots m^{n_m})$ 的权 $w(\pi)$ 定义为在两行形式下它的列的权的乘积，其中 ^j_k 的权当 j 和 k 两者都 $\leqslant t$ 或 $>t$ 时为 w_j/w_k，否则为 z_j/z_k。试证明对所有的 $\pi\in P(1^{n_1}\cdots m^{n_m})$，$w(\pi)$ 之和为：

$$\sum_{k\geqslant 0}\frac{k!^2(n_{\leqslant t}-k)!(n_{>t}-k)!}{n_1!\cdots n_m!}\left|\sum\binom{n_1}{p_1}\cdots\binom{n_m}{p_m}\left(\frac{w_1}{z_1}\right)^{p_1}\cdots\left(\frac{w_m}{z_m}\right)^{p_m}\right|^2$$

其中 $n_{\leqslant t}$ 是 $n_1+\cdots+n_t$，$n_{>t}$ 是 $n_{t+1}+\cdots+n_m$，而且内部的求和是对于使得 $p_{\leqslant t}=p_{>t}=k$ 的所有

$(p_1,\cdots,p_m)$进行的。

23.[*M23*] 一股脱氧核糖核酸(DNA)可以想像为在一个 4 个字母的字母表上的一个字。假设我们复制一股脱氧核糖核酸并且把它完全分裂成单字母基,然后再随机地重新组合它们。如果把得到的这股放在原来一股之后。试证它们不同的位置个数相对奇数来说更有可能是偶数[提示:应用上一道习题]。

24.[*27*] 考虑在两个未排序的字母偶之间可能成立的任何关系 R;如果$\{w,x\}R\{y,z\}$,我们就说$\{w,x\}$保持$\{y,z\}$,否则就说$\{w,x\}$移动$\{y,z\}$。

假定 $w\neq x$ 和 $y\neq z$,按照对偶$\{w,x\}$保持还是移动对偶$\{y,z\}$,对于 R,转置$\begin{smallmatrix}w & x\\ y & z\end{smallmatrix}$的操作为以$\begin{smallmatrix}x & w\\ y & z\end{smallmatrix}$或$\begin{smallmatrix}x & w\\ z & y\end{smallmatrix}$来取代$\begin{smallmatrix}w & x\\ y & z\end{smallmatrix}$。

对于 R 来说,对一个两行数组$\begin{pmatrix}x_1\cdots x_n\\ y_1\cdots y_n\end{pmatrix}$的排序操作为重复地求最大的 x_j,并且若 $x_j>x_{j+1}$ 则转置列 j 和 $j+1$,直到最终有 $x_1\leqslant\cdots\leqslant x_n$ 为止(我们不要求 $y_1\cdots y_n$ 是 $x_1\cdots x_n$ 的一个排列)。

a)给定$\begin{pmatrix}x_1\cdots & x_n\\ y_1\cdots & y_n\end{pmatrix}$,试证明,对于每个 $x\in\{x_1,\cdots,x_n\}$,存在惟一的 $y\in\{y_1,\cdots,y_n\}$,使得对于某个 $x_2',y_2',\cdots,x_n',y_n'$,$\text{sort}\begin{pmatrix}x_1\cdots x_n\\ y_1\cdots y_n\end{pmatrix}=\text{sort}\begin{pmatrix}x\ x_2'\cdots x_n'\\ y\ y_2'\cdots y_n'\end{pmatrix}$。

b)令$\begin{pmatrix}w_1\cdots w_k\\ y_1\cdots y_k\end{pmatrix}$Ⓡ$\begin{pmatrix}x_1\cdots x_l\\ z_1\cdots z_l\end{pmatrix}$表示相对于 R 对$\begin{pmatrix}w_1\cdots w_k & x_1\cdots x_l\\ y_1\cdots y_k & z_1\cdots z_l\end{pmatrix}$进行排序的结果。例如,如果 R 总是为真,则Ⓡ只不过是连接;如果 R 总为假,则Ⓡ是交插乘积 ⊤。如果我们令(11)表示$(x_2\cdots\ x_nx_1)$而不是$(x_1x_2\cdots\ x_n)$来重新定义轮换符号,则通过证明一个多重集合 M 的每一个排列,有满足(16)的形如

$$\pi=(x_{11}\cdots\ x_{1n_1}y_1)\text{Ⓡ}((x_{21}\cdots\ x_{2n_2}y_2)\text{Ⓡ}\ \cdots\text{Ⓡ}(x_{t1}\cdots\ x_{tn_t}y_t))$$

的惟一表示,可以得出定理 A。例如,假设$\{w,x\}R\{y,z\}$意味着 w,x,y 和 z 都不同,则类似于(17)的(12)的因子分解为

$$(d\ d\ b\ c\ a)\text{Ⓡ}((c\ b\ b\ a)\text{Ⓡ}((c\ d\ b)\text{Ⓡ}((d\ b)\text{Ⓡ}(d))))$$

(操作Ⓡ不总是遵从结合律;在广义的因子分解中的括弧应从右至左地嵌套。)

*5.1.3 路段

在第 3 章中,我们分析了排列中上运行的长度,作为测试一个序列的随机性的一项方法。如果在一个排列 $a_1\ a_2\cdots\ a_n$ 的两端各放一根垂直的线段,而且每当 $a_j>a_{j+1}$时,也在 a_j 与 a_{j+1}之间放一根直线,则路段(runs)就是诸对线之间的区段(segments)。例如,排列

$$|\,3\ 5\ 7\,|\,1\ 6\ 8\ 9\,|\,4\,|\,2\,|$$

有 4 个路段。在 3.3.2G 小节中建立的理论,确定了在$\{1,2,\cdots,n\}$的一个随机排列中,长度为 k 的路段的平均个数,以及长度为 j 和 k 的路段个数的协方差。路段在排序算法的研究中是重要的,因为它们表示数据已排序的诸段,所以现在将再次研究路段这个课题。

让我们使用记号

$$\left\langle {n \atop k} \right\rangle \tag{1}$$

来代表$\{1,2,\cdots,n\}$的排列中，恰有 k 个"递减"$a_j > a_{j+1}$，因而恰有 $k+1$ 个递增路段的个数。数$\left\langle {n \atop k} \right\rangle$在许多场合中出现，它们通常被称做欧拉，因为欧拉在很多年之前的一篇技术文章[*Comment. Acad. Sci. Imp. Petrop.* **8**(1736)，147～158，§13]中提到了它们，之后又在他的名著[*Institutiones Calculi Differentialis* (St. Petersburg:1755)，485～487]中讨论了它们。注意，不要把它们与习题 5.1.4-23 中讨论的"欧拉(Euler)数"E_n 相混淆。$\left\langle {n \atop k} \right\rangle$中的尖括号提醒我们在一个递降定义中的">"符号。当然$\left\langle {n \atop k} \right\rangle$也是有 k 个"递增"的 $a_j < a_{j+1}$ 的排列个数。

通过把元素 n 插入到所有可能的位置上，我们可以使用对$\{1,\cdots,n-1\}$的任何给定的排列来形成 n 个新的排列。如果原来的排列有 k 个递降则恰有$k+1$ 个新的排列将有 k 个递降；剩下的 $n-1-k$ 个将有$k+1$，因为除非把元素 n 放在一个现存路段的末尾，否则我们就会增加递降的数目。例如，从 3 1 2 4 5 形成的 6 个排列是

6 3 1 2 4 5，　3 6 1 2 4 5，　3 1 6 2 4 5，
3 1 2 6 4 5，　3 1 2 4 6 5，　3 1 2 4 5 6；

除第二个和最后一个外，它们都有两个递降，而不是一个，因此有递推关系

$$\left\langle {n \atop k} \right\rangle = (k+1)\left\langle {n-1 \atop k} \right\rangle + (n-k)\left\langle {n-1 \atop k-1} \right\rangle, \text{整数 } n > 0, \text{整数 } k \tag{2}$$

按约定，置

$$\left\langle {0 \atop k} \right\rangle = \delta_{k0} \tag{3}$$

指出，空排列没有递降。读者可以发现，把(2)同在等式 1.2.6-(46)中的斯特林数的递推关系作比较是有趣的，表 1 列出了对于小的 n 的欧拉数。

表 1　欧拉数

n	$\left\langle {n \atop 0} \right\rangle$	$\left\langle {n \atop 1} \right\rangle$	$\left\langle {n \atop 2} \right\rangle$	$\left\langle {n \atop 3} \right\rangle$	$\left\langle {n \atop 4} \right\rangle$	$\left\langle {n \atop 5} \right\rangle$	$\left\langle {n \atop 6} \right\rangle$	$\left\langle {n \atop 7} \right\rangle$	$\left\langle {n \atop 8} \right\rangle$
0	1	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0	0
3	1	4	1	0	0	0	0	0	0
4	1	11	11	1	0	0	0	0	0
5	1	26	66	26	1	0	0	0	0
6	1	57	302	302	57	1	0	0	0
7	1	120	1191	2416	1191	120	1	0	0

（续）

n	$\left\langle {n \atop 0} \right\rangle$	$\left\langle {n \atop 1} \right\rangle$	$\left\langle {n \atop 2} \right\rangle$	$\left\langle {n \atop 3} \right\rangle$	$\left\langle {n \atop 4} \right\rangle$	$\left\langle {n \atop 5} \right\rangle$	$\left\langle {n \atop 6} \right\rangle$	$\left\langle {n \atop 7} \right\rangle$	$\left\langle {n \atop 8} \right\rangle$
8	1	247	4293	15619	15619	4293	247	1	0
9	1	502	14608	88234	156190	88234	14608	502	1

在表 1 中可以观察到若干模式。由定义，我们有

$$\left\langle {n \atop 0} \right\rangle + \left\langle {n \atop 1} \right\rangle + \cdots + \left\langle {n \atop n} \right\rangle = n! \tag{4}$$

$$\left\langle {n \atop 0} \right\rangle = 1 \tag{5}$$

$$\left\langle {n \atop n-1} \right\rangle = 1, \quad \left\langle {n \atop n} \right\rangle = 0, \text{对于 } n \geqslant 1 \tag{6}$$

由一般的对称规则，等式(6)由等式(5)得出，这是因为由一般的对称性规则，

$$\left\langle {n \atop k} \right\rangle = \left\langle {n \atop n-1-k} \right\rangle, \text{对于 } n \geqslant 1 \tag{7}$$

这由下列事实得出，即有 k 个递降的每个非空的排列 $a_1 a_2 \cdots a_n$ 有 $n-1-k$ 个递增。欧拉数另一个重要的性质是公式

$$\sum_k \left\langle {n \atop k} \right\rangle \binom{m+k}{n} = m^n, \quad n \geqslant 0 \tag{8}$$

它由中国的数学家 Li Shan-Lan(李善兰)发现并发表于 1867 年[参见 J.-C. Martzloff, *A History of Chinese Mathematics* (Berlin: Springer, 1997), 346～348；对于 $n \leqslant 5$ 的特殊情况，已经由日本的 Yoshisuke Matsunaga(松永良弼)所知晓，他卒于 1744 年]。李善兰的恒等式是根据排序的性质推出的：考虑使得 $1 \leqslant a_i \leqslant m$ 的 m^n 个序列 $a_1 a_2 \cdots a_n$。我们可以以稳定的方式把任何这样的序列排成非递减的次序，得到

$$a_{i_1} \leqslant a_{i_2} \leqslant \cdots \leqslant a_{i_n} \tag{9}$$

其中 $i_1 i_2 \cdots i_n$ 是 $\{1, 2, \cdots, n\}$ 的一个惟一确定的排列，使得 $a_{i_j} = a_{i_{j+1}}$ 意味着 $i_j < i_{j+1}$；换言之，$i_j > i_{j+1}$ 意味着 $a_{i_j} < a_{i_{j+1}}$。如果排列 $i_1 i_2 \cdots i_n$ 有 k 个路段，则将证明对应的序列 $a_1 a_2 \cdots a_n$ 的个数是 $\binom{m+n-k}{n}$。如果我们以 $n-k$ 来代替 k 并且使用(7)，就将证明(8)，因为 $\left\langle {n \atop k} \right\rangle$ 个排列有 $n-k$ 个路段。

例如，如果 $n=9$ 且 $i_1 i_2 \cdots i_n = 3\ 5\ 7\ 1\ 6\ 8\ 9\ 4\ 2$，则我们要计算使得

$$1 \leqslant a_3 \leqslant a_5 \leqslant a_7 < a_1 \leqslant a_6 \leqslant a_8 \leqslant a_9 < a_4 < a_2 \leqslant m \tag{10}$$

的序列 $a_1 a_2 \cdots a_n$ 的个数，是使得

$$1 \leqslant b_1 < b_2 < b_3 < b_4 < b_5 < b_6 < b_7 < b_8 < b_9 \leqslant m+5$$

的序列 $b_1b_2\cdots b_9$ 的个数，因为我们能令 $b_1=a_3, b_2=a_5+1, b_3=a_7+2, b_4=a_1+2, b_5=a_6+3$，等等。选择诸 b 的方式的个数，就是由 $m+5$ 件事物中选择 9 件的方式数，即 $\binom{m+5}{9}$；类似的证明对于一般的 k 和 n，以及对于任何具有 k 个路段的任何排列 $i_1\ i_2\cdots\ i_n$ 也有效。

由于(8)的两边都是 m 的多项式，故可以用任何实数 x 来代替 m，从而得到用相继的二项式系数表示乘方的一种有趣方式：

$$x^n=\left\langle {n \atop 0} \right\rangle\binom{x}{n}+\left\langle {n \atop 1} \right\rangle\binom{x+1}{n}+\cdots+\left\langle {n \atop n-1} \right\rangle\binom{x+n-1}{n},\quad n\geqslant 1 \tag{11}$$

例如，

$$x^3=\binom{x}{3}+4\binom{x+1}{3}+\binom{x+2}{3}$$

这是欧拉数的键码性质，这种性质使得欧拉数在离散数学的研究中很有用。

在(11)中置 $x=1$，就再次证明了

$$\left\langle {n \atop n-1} \right\rangle=1$$

因为二项式系数除了最后一项外都消失了。置 $x=2$，得到

$$\left\langle {n \atop n-2} \right\rangle=\left\langle {n \atop 1} \right\rangle=2^n-n-1,\quad n\geqslant 1 \tag{12}$$

置 $x=3,4,\cdots$ 即知关系(11)完全确定了数 $\left\langle {n \atop k} \right\rangle$，并且导出了最初由欧拉给出的一个公式：

$$\left\langle {n \atop k} \right\rangle=(k+1)^n-k^n\binom{n+1}{1}+(k-1)^n\binom{n+1}{2}-\cdots+(-1)^k1^n\binom{n+1}{k}=$$
$$\sum_{j=0}^{k}(-1)^j\binom{n+1}{j}(k+1-j)^n,\quad n\geqslant 0,k\geqslant 0 \tag{13}$$

现在研究路段的生成函数。如果置

$$g_n(z)=\sum_k\left\langle {n \atop k-1} \right\rangle\frac{z^k}{n!} \tag{14}$$

则 z^k 的系数是 $\{1,2,\cdots,n\}$ 的一个随机排列恰有 k 个路段的概率。由于 k 个路段的可能性恰巧等同于 $n+1-k$ 个，故路段的平均数必须是 $\frac{1}{2}(n+1)$，因此 $g_n'(1)=\frac{1}{2}(n+1)$。习题 2(b)证明，对于 $g_n(z)$ 在点 $z=1$ 的各次导数，有一个简单的公式

$$g_n^{(m)}(1)=\left\{ {n+1 \atop n+1-m} \right\}\Big/\binom{n}{m},\quad n\geqslant m \tag{15}$$

因此特别地，对于 $n\geqslant 2$，方差 $g''_n(1)+g'_n(1)-g'_n(1)^2$ 成为 $(n+1)/12$，表示对于

均值的稍微稳定的分布(我们在等式 3.3.2-(18)中发现过同样的量,在那里它被称做协方差(R'_1,R'_1))。由于 $g_n(z)$是一个多项式,我们可以使用公式(15)推导出泰勒级数展开式

$$g_n(z)=\frac{1}{n!}\sum_{k=0}^{n}(z-1)^{n-k}k!\left\{{n+1 \atop k+1}\right\}=$$

$$\frac{1}{n!}\sum_{k=0}^{n}z^{k+1}(1-z)^{n-k}k!\left\{{n+1 \atop k+1}\right\} \tag{16}$$

此公式的第二式从第一式得出,因为由对称性条件(7)

$$g_n(z)=z^{n+1}g_n\left(\frac{1}{z}\right),\quad n\geqslant 1 \tag{17}$$

斯特林数递归式

$$\left\{{n+1 \atop k+1}\right\}=(k+1)\left\{{n \atop k+1}\right\}+\left\{{n \atop k}\right\}$$

给出了两个稍微更简单些的表示,当 $n\geqslant 1$ 时,

$$g_n(z)=\frac{1}{n!}\sum_{k=0}^{n}z(z-1)^{n-k}k!\left\{{n \atop k}\right\}=$$

$$\frac{1}{n!}\sum_{k=0}^{n}z^{k}(1-z)^{n-k}k!\left\{{n \atop k}\right\} \tag{18}$$

因此,超生成函数

$$g(z,x)=\sum_{n\geqslant 0}\frac{g_n(z)x^n}{z}=\sum_{k,n\geqslant 0}\left\langle{n \atop k}\right\rangle\frac{z^k x^n}{n!} \tag{19}$$

等于

$$\sum_{k,n\geqslant 0}\frac{((z-1)x)^n}{(z-1)^k}\left\{{n \atop k}\right\}\frac{k!}{n!}=\sum_{k\geqslant 0}\left(\frac{e^{(z-1)x}-1}{z-1}\right)^k=\frac{(1-z)}{e^{(z-1)x}-z} \tag{20}$$

这是欧拉讨论的另一个关系。

欧拉数进一步的性质可以在 L. Carlitz 的综述性论文[*Math. Magazine* **33** (1959), 247 ~ 260]中找到;也可见 J. Riordan 的 *Introduction to Combinatorial Analysis* (New York: Wiley, 1958), 38 ~ 39, 214 ~ 219, 234 ~ 237; D. Foata 和 M. P. Schützenberger 的 *Lecture Notes in Math.* **138** (Berlin: Springer, 1970)。

现在考虑路段的长度;平均说来,一个路段的长度等于多少?在 3.3.2 小节,已经研究了给定长度路段的预期数目;路段的平均长度近似为 2,同长度为 n 的一个随机排列中大约有 $\frac{1}{2}(n+1)$个路段这一事实相一致。为了应用于排序算法,换一个角度来考虑是有用的;对于 $k=1,2,\cdots$,考虑在排列中从左到右第 k 个路段的长度。

例如,一个随机排列 $a_1a_2\cdots a_n$ 的头一个(最左边的)路段有多长?它的长度总是$\geqslant 1$,而且恰有一半的次数,它的长度$\geqslant 2$(即当 $a_1<a_2$ 时)。恰有 1/6 的次数,它

的长度≥3(当 $a_1<a_2<a_3$ 时)。而且一般说来,对于 $1\leqslant m\leqslant n$,它的长度≥m 的概率是 $q_m=1/m!$。因此,它的长度恰恰等于 m 的概率是

$$p_m = q_m - q_{m+1} = 1/m! - 1/(m+1)!,\quad \text{对于 } 1\leqslant m < n$$

$$p_n = 1/n! \tag{21}$$

因此头一个路段的平均长度为

$$p_1 + 2p_2 + \cdots + np_n = (q_1 - q_2) + 2(q_2 - q_3) + \cdots +$$

$$(n-1)(q_{n-1} - q_n) + nq_n = q_1 + q_2 + \cdots + q_n = \frac{1}{1!} + \frac{1}{2!} + \cdots + \frac{1}{n!} \tag{22}$$

如果令 $n\to\infty$,则极限为 $e-1=1.71828\cdots$,而且对于有限的 n,这个值为 $e-1-\delta_n$,其中 δ_n 十分小

$$\delta_n = \frac{1}{(n+1)!}\left(1 + \frac{1}{n+2} + \frac{1}{(n+2)(n+3)} + \cdots\right) \leqslant \frac{e-1}{(n+1)!}$$

因此,为了实用,不妨研究在不同数的随机无穷序列

$$a_1, a_2, a_3, \cdots$$

中的路段;这里我们所说的“随机”,是指在这个序列中,头 n 个元素的 $n!$ 种相对次序的每一种都是同等可能的。因此,在一个随机无穷序列中头一个路段的平均长度为 $e-1$。

只要稍微地加强对头一个路段的分析,就能确认在一个随机序列中第 k 个路段的平均长度。设 q_{km} 是头 k 个路段的总长度≥m 的概率,则 q_{km} 是 $1/m!$ 乘以路段数 $\leqslant k$ 的 $\{1,2,\cdots,m\}$ 的排列数

$$q_{km} = \left(\left\langle {m \atop 0} \right\rangle + \cdots + \left\langle {m \atop k-1} \right\rangle\right)\Big/ m! \tag{23}$$

头 k 个路段总长度是 m 的概率为 $q_{km}-q_{k(m+1)}$。因此,如果 L_k 表示第 k 个路段的平均长度,则求得

$$\begin{aligned}&L_1 + \cdots + L_k = \text{头 } k \text{ 个路段的平均总长度} =\\ &(q_{k1} - q_{k2}) + 2(q_{k2} - q_{k3}) + 3(q_{k3} - q_{k4}) + \cdots =\\ &q_{k1} + q_{k2} + q_{k3} + \cdots\end{aligned}$$

减去 $L_1+\cdots+L_{k-1}$ 并利用(23)中的 q_{km} 的值,即得所求的公式

$$L_k = \frac{1}{1!}\left\langle {1 \atop k-1} \right\rangle + \frac{1}{2!}\left\langle {2 \atop k-1} \right\rangle + \frac{1}{3!}\left\langle {3 \atop k-1} \right\rangle + \cdots = \sum_{m\geqslant 1}\left\langle {m \atop k-1} \right\rangle \frac{1}{m!} \tag{24}$$

由于除当 $k=1$ 外 $\left\langle {0 \atop k-1} \right\rangle =0$，故 L_k 可证明是生成函数 $g(z,1)-1$(见等式(19))中 z^{k-1} 的系数，所以有

$$L(z) = \sum_{k\geqslant 0} L_k z^k = \frac{z(1-z)}{e^{z-1}-z} - z \tag{25}$$

由欧拉公式(13)得到 L_k 作为 e 的一个多项式的表示

$$\begin{aligned} L_k &= \sum_{m\geqslant 0}\sum_{j=0}^{k}(-1)^{k-j}\binom{m+1}{k-j}\frac{j^m}{m!} = \\ &\sum_{j=0}^{k}(-1)^{k-j}\sum_{m\geqslant 0}\binom{m}{k-j}\frac{j^m}{m!} + \sum_{j=0}^{k}(-1)^{k-j}\sum_{m\geqslant 0}\binom{m}{k-j-1}\frac{j^m}{m!} = \\ &\sum_{j=0}^{k}\frac{(-1)^{k-j}j^{k-j}}{(k-j)!}\sum_{n\geqslant 0}\frac{j^n}{n!} + \sum_{j=0}^{k}\frac{(-1)^{k-j}j^{k-j-1}}{(k-j-1)!}\sum_{n\geqslant 0}\frac{j^n}{n!} = \\ &k\sum_{j=0}^{k}\frac{(-1)^{k-j}j^{k-j-1}}{(k-j)!}e^j \end{aligned} \tag{26}$$

这个 L_k 的公式首先由 B. J. Gassner 得到[见*CACM* **10** (1967)，89～93]。特别是，有

$$L_1 = e-1 \approx 1.71828\cdots$$

$$L_2 = e^2-2e \approx 1.95249\cdots$$

$$L_3 = e^3-3e^2+\frac{3}{2}e \approx 1.99579\cdots$$

所以预期第二个路段较头一个长，而且平均地说，第三个还将更长！乍一看，这似乎令人惊奇，但稍经思考即可知，由于第二个路段的头一个元素趋于变小(它引起第一个路段终止)，故第二个路段就有一个较好的机会变得更长些。第三个路段的头一个元素甚至比第二个的更小。

在替代-选择排序理论中，数 L_k 有其重要性(5.4.1 小节)，所以详细研究它们的值是有意义的。表 2 列出了 L_k 的头 18 个值到 15 位十进数。在上段中的讨论可能使我们首先猜测 $L_{k+1}>L_k$，但是事实上这些值是向后和向前摆动的。注意，L_k 迅速地趋于极限值 2；看看超越数 e 的这些单一多项式多么迅速地收敛到有理数 2 是很有意思的！从数值分析的观点来看，多项式(26)也是颇为有趣的，因为它们提供了当接近相等的数相减时有效位数丢失的一个精采的例子；利用 19 位浮点算术，Gassner 得到不正确的结论 $L_{12}>2$，而 John W. Wrench，Jr. 注意到，用 42 位浮点算术来计算 L_{28} 时仅仅正确到 29 位有效数字。

表 2 第 k 个路段的平均长度

k	L_k	k	L_k
1	1.71828 18284 59045 +	10	2.00000 00012 05997 +
2	1.95249 24420 12560 −	11	2.00000 00001 93672 +
3	1.99579 13690 84285 −	12	1.99999 99999 99909 +
4	2.00003 88504 76806 −	13	1.99999 99999 97022 −
5	2.00005 75785 89716 +	14	1.99999 99999 99719 +
6	2.00000 50727 55710 −	15	2.00000 00000 00019 +
7	1.99999 96401 44022 +	16	2.00000 00000 00006 +
8	1.99999 98889 04744 +	17	2.00000 00000 00000 +
9	1.99999 99948 43434 −	18	2.00000 00000 00000 −

L_k 的渐近性质可用复变数理论的简单原理来确定。仅当 $e^{z-1}=z$，即如果我们写 $z=x+iy$，当

$$e^{x-1}\cos y = x \quad 和 \quad e^{x-1}\sin y = y \tag{27}$$

时，(25)的分母为 0。图 3 所示为这两个方程的叠印图。我们注意到，它们在 $z=z_0, z_1, \bar{z}_1, z_2, \bar{z}_2, \cdots$ 处相交，这里，$z_0=1$，

$$z_1 = (3.08884\ 30156\ 13044\ -) + (7.46148\ 92856\ 54255\ -)i \tag{28}$$

而且虚部 $\Im(z_{k+1})$ 对于很大的 k 粗略地等于 $\Im(z_k)+2\pi$。由于

$$\lim_{z\to z_k}\left(\frac{1-z}{e^{z-1}-z}\right)(z-z_k) = -1, \quad 对于\ k>0$$

且对 $k=0$，此极限为 -2，故函数

$$R_m(z) = L(z) + \frac{2}{z-z_0} + \frac{z_1}{z-z_1} + \frac{\bar{z}_1}{z-\bar{z}_1} + \frac{z_2}{z-z_2} + \frac{\bar{z}_2}{z-\bar{z}_2} + \cdots + \frac{z_m}{z-z_m} + \frac{\bar{z}_m}{z-\bar{z}_m}$$

在 $|z|<|z_{m+1}|$ 的复平面中没有奇异点。因此 $R_m(z)$ 有一个幂级数展开式 $\sum_k \rho_k z^k$，当 $|z|<|z_{m+1}|$ 时，它绝对收敛；由此得出当 $k\to\infty$ 时，$\rho_k M^k \to 0$，其中 $M=|z_{m+1}|-\epsilon$。$L(z)$ 的系数是

$$\frac{2}{1-z} + \frac{1}{1-z/z_1} + \frac{1}{1-z/\bar{z}_1} + \cdots + \frac{1}{1-z/z_m} + \frac{1}{1-z/\bar{z}_m} + R_m(z)$$

的系数，即

$$L_n = 2 + 2r_1^{-n}\cos n\theta_1 + 2r_2^{-n}\cos n\theta_2 + \cdots + 2r_m^{-n}\cos n\theta_m + O(r_{m+1}^{-n}) \tag{29}$$

若令

$$z_k = r_k e^{i\theta_k} \tag{30}$$

这说明了 L_n 的渐近性质。我们有

$$\begin{aligned} r_1 &= 8.07556\ 64528\ 89526-, & \theta_1 &= 1.17830\ 39784\ 74668+ \\ r_2 &= 14.35456\ 68997\ 62106-, & \theta_2 &= 1.31268\ 53883\ 87636+ \\ r_3 &= 20.62073\ 15381\ 80628-, & \theta_3 &= 1.37427\ 90757\ 91688- \\ r_4 &= 26.88795\ 29424\ 54546-, & \theta_4 &= 1.41049\ 72786\ 51865- \end{aligned} \tag{31}$$

所以对于 L_n-2 的主要贡献乃归功于 r_1 和 θ_1，而且(29)的收敛是十分迅速的。进一步的分析[W. W. Hooker, *CACM* **12** (1969), 411～413]证明，当 $m\to\infty$ 时，$R_m(z)\to -z$；因此当 $n>1$ 时，级数 $2\sum_{k\geq 0} r_k^{-n}\cos n\theta_k$ 实际上收敛于 L_n。

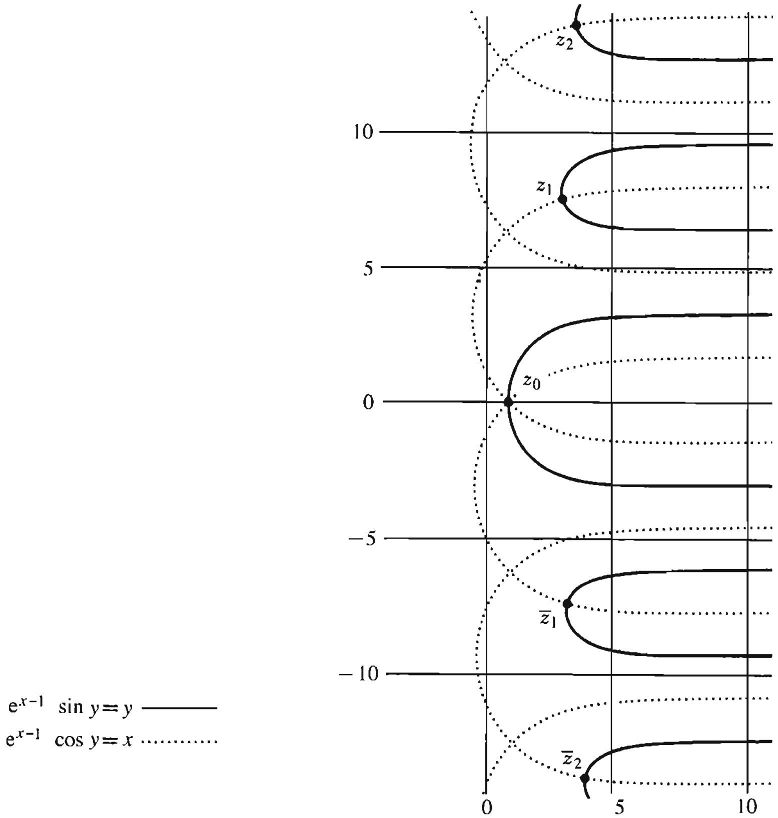


图 3　$e^{z-1}=z$ 的根

可以对概率进行更仔细的考察，以确定对于第 k 个路段长度和对头 k 个路段的总长度的整个概率分布(见习题 9,10,11)。和数 $L_1+L_2+\cdots+L_k$ 结果是渐近于 $2k-\frac{1}{3}+O(8^{-k})$。

让我们来考虑，当在排列中允许出现相等的元素时路段的性质，以此来结束本

小节。19 世纪著名的美国天文学家 Simon Newcomb 对做一种与这个问题有关的单人游戏很感兴趣。他把一副扑克牌一张一张地放在桌上,只要牌的面值是以非减次序出现的,他就把它们堆成一堆;但每当待堆放的下一张牌的面值小于前者时,他就开始一个新的堆。他想要知道,在整副牌以这种方式分发完毕之后,形成给定堆数的概率。

因此,Simon Newcomb 的问题就是,求在一个多重集合的随机排列中,路段的概率分布问题。尽管已经看过怎样来解决当所有的牌都有不同的面值时的特殊情况,但一般的回答是比较复杂的(见习题 12)。这里,我们将满足于推导出现于这个游戏中的平均堆数。

首先假定有 m 种不同类型的牌,每种类型恰出现 p 次。例如,通常的桥牌,如果不考虑花色时,有 $m=13$ 和 $p=4$。P. A. MacMahon 发现了应用于这种情况的值得注意的对称性[*Combinatory Analysis* **1** (Cambridge,1915),212～213]:有 $k+1$ 个路段的排列个数等同于有 $mp-p-k+1$ 个路段的排列个数。当 $p=1$ 时,这个关系是等式(7),但对于 $p>1$ 时,它是十分令人惊奇的。

通过在具有 $k+1$ 个路段的每个排列,同具有 $mp-p-k+1$ 个路段的另一个排列之间建立一一对应,就能够证明这对称性。在往下阅读之前,鼓励读者亲自动手试一试,来发现这一对应。

显然,没有很简单的对应;MacMahon 的证明是以生成函数,而不是以一种组合构造为基础的。但 Foata 的对应(定理 5.1.2B)提供了一个有用的简化,因为它告诉我们,在具有 $k+1$ 个路段的多重集合排列和其两行表示法中,对于 $x<y$,恰含有 k 个列^y_x 的排列之间,有一一对应。

假设给定的多重集合是 $\{p\cdot 1, p\cdot 2,\cdots, p\cdot m\}$,考察其两行表示法为

$$\begin{pmatrix} 1 & \cdots & 1 & 2 & \cdots & 2 & \cdots & m & \cdots & m \\ x_{11} & \cdots & x_{1p} & x_{21} & \cdots & x_{2p} & \cdots & x_{m1} & \cdots & x_{mp} \end{pmatrix} \tag{32}$$

的排列,可以把这个排列和同一多重集合的另一个排列结合起来

$$\begin{pmatrix} 1 & \cdots & 1 & 2 & \cdots & 2 & \cdots & m & \cdots & m \\ x'_{11} & \cdots & x'_{1p} & x'_{m1} & \cdots & x'_{mp} & \cdots & x'_{21} & \cdots & x'_{2p} \end{pmatrix} \tag{33}$$

其中 $x'=m+1-x$。如果(32)含有形如^y_x 且 $x<y$ 的 k 个列,则(33)含有 $(m-1)p-k$ 个这样的列;因为仅仅需要考虑 $y>1$ 的情况,而且 $x<y$ 等价于 $x'\geqslant m+2-y$。现在(32)对应于具有 $k+1$ 个路段的一个排列,而(33)对应于具有 $mp-p-k+1$ 个路段的一个排列,而且由于把(32)变为(33)的变换是可逆的——它把(33)变回(32),因此 MacMahon 的对称性条件已经建立。关于这个构造的一个例子,见习题 14。

由于对称性,在一个随机排列中路段的平均数必须是 $\frac{1}{2}((k+1)+(mp-p-k+1))=1+\frac{1}{2}p(m-1)$。例如,利用一副标准扑克牌,由 Simon Newcomb 的单人游戏得到的平均堆数将是 25(所以它不像是一种有吸引力的单人游戏)。

给定任意多重集合$\{n_1\cdot x_1, n_2\cdot x_2, \cdots, n_m\cdot x_m\}$，其中诸 x 是不同的，利用一个相当简单的论证，实际上可以大体上确定路段的平均数。设 $n=n_1+n_2+\cdots+n_m$，并想像这个多重集合的所有排列 $a_1\,a_2\cdots\,a_n$ 都已经写下来；我们来考察，对于每个固定的 i 值 $1\leqslant i<n$，a_i 大于a_{i+1}的机会如何。$a_i>a_{i+1}$的次数恰是 $a_i\neq a_{i+1}$的次数之半；而且不难看出，$a_i=a_{i+1}=x_j$ 恰有 $Nn_j(n_j-1)/n(n-1)$次，其中 N 是排列的总数。因此，$a_i=a_{i+1}$恰有

$$\frac{N}{n(n-1)}(n_1(n_1-1)+\cdots+n_m(n_m-1))=\frac{N}{n(n-1)}(n_1^2+\cdots+n_m^2-n)$$

次，而且 $a_i>a_{i+1}$恰有

$$\frac{N}{2n(n-1)}(n^2-(n_1^2+\cdots+n_m^2))$$

次。因为在每个排列中有一个路段在 a_n 处结束，因此若对 i 求和并加上N，则得到在所有 N 个排列当中路段的总数

$$N\left(\frac{n}{2}-\frac{1}{2n}(n_1^2+\cdots+n_m^2)+1\right) \tag{34}$$

除以 N 即给出所求的平均路段数。

由于在“顺序统计”的研究中路段是重要的，因此有相当多的著作讨论它们，包括这里未予考虑的若干其它类型的路段在内。更多相关信息，请参见 F.N.David 和 D.E.Barton 所著的书*Combinatorial Chance* (London:Griffin,1962)，第 10 章；以及由 D.E.Barton 与 C.L.Mallows 合写的综述论文*Annals of Math. Statistics* **36** (1965)，236～260。

习　题

1.[*M26*] 推导欧拉公式(13)。

▶2.[*M22*] (a)推广正文中用来证明(8)的思想，考虑恰含 q 个不同元素的那些序列$a_1a_2\cdots a_n$，以证明

$$\sum_k \left\langle {n \atop k} \right\rangle \binom{k}{n-q} = \left\{ {n \atop q} \right\} q!$$

(b)用这一恒等式来证明，对于 $n\geqslant m$，

$$\sum_k \left\langle {n \atop k} \right\rangle \binom{k+1}{m} = \left\{ {n+1 \atop n+1-m} \right\} (n-m)!$$

3.[*HM25*] 计算和数 $\sum_k \left\langle {n \atop k} \right\rangle (-1)^k$。

4.[*M21*] $\sum_k (-1)^k \left\{ {n \atop k} \right\} k! \binom{n-k}{m}$的值等于多少？

5.[*M20*] 当 p 是素数时，求 $\left\langle {p \atop k} \right\rangle \bmod p$ 的值。

▶6.[*M21*] B.C.Dull 先生注意到，由等式(4)和(13)

$$n! = \sum_{k\geqslant 0}\left\langle {n \atop k} \right\rangle = \sum_{k\geqslant 0}\sum_{j\geqslant 0}(-1)^{k-j}\binom{n+1}{k-j}(j+1)^n$$

首先对 k 进行求和,他发现对于所有的 $j\geqslant 0$, $\sum_{k\geqslant 0}(-1)^{k-j}\binom{n+1}{k-j}=0$;因此,对于所有的 $n\geqslant 0$, $n!=0$,他是否正确?

7.[*HM40*] 由(14)给出的路段的概率分布,是否渐近于正态的(参见 1.2.10-13)?

8.[*M24*] (P.A.MacMahon) 证明一个充分长的排列,其第一个路段的长度为 l_1,第二个的长度为 l_2,…,以及第 k 个的长度 $\geqslant l_k$ 的概率,是

$$\det\begin{pmatrix} 1/l_1! & 1/(l_1+l_2)! & 1/(l_1+l_2+l_3)! & \cdots & 1/(l_1+l_2+l_3+\cdots+l_k)! \\ 1 & 1/l_2! & 1/(l_2+l_3)! & \cdots & 1/(l_2+l_3+\cdots+l_k)! \\ 0 & 1 & 1/l_3! & \cdots & 1/(l_3+\cdots+l_k)! \\ \vdots & & & & \vdots \\ 0 & 0 & \cdots & 1 & 1/l_k! \end{pmatrix}$$

9.[*M30*] 设 $h_k(z)=\sum p_{km}z^m$,其中 p_{km} 是在一个随机(无穷)序列中头 k 个路段的总长度为 m 的概率。试求 $h_1(z)$,$h_2(z)$以及超生成函数 $h(z,x)=\sum_k h_k(z)x^k$ 的"简单"表达式。

10.[*HM30*] 对于大的 k,求上题中的分布 $h_k(z)$的均值和方差的渐近特性。

11.[*M40*] 设 $H_k(z)=\sum P_{km}z^m$,其中 P_{km}是在一个随机(无穷)序列中第 k 个路段的长度为 m 的概率。试借助熟悉的函数来表达 $H_1(z)$,$H_2(z)$以及超生成函数$H(z,x)=\sum_k H_k(z)x^k$ 。

12.[*M33*] (P.A.MocMahon)通过证明恰有 k 个路段的$\{n_1\cdot 1,n_2\cdot 2,\cdots,n_m\cdot m\}$的排列数是

$$\sum_{j=0}^{k}(-1)^j\binom{n+1}{j}\binom{n_1-1+k-j}{n_1}\binom{n_2-1+k-j}{n_2}\cdots\binom{n_m-1+k-j}{n_m}$$

其中,$n=n_1+n_2+\cdots+n_m$,把等式(13)推广到一个多重集合的排列上。

13.[*05*] 如果 Simon Newcomb 的单人游戏用一副标准桥牌来进行,忽略面值,但令梅花<方片<红桃<黑桃,问平均的堆数等于多少?

14.[M*18*] 排列 3 1 1 1 2 3 1 4 2 3 3 4 2 2 4 4 有 5 个路段;试按照正文关于 MacMahon 对称性条件的构造,求对应的有 9 个路段的排列。

▶15.[M*21*](交错路段) 19 世纪有关组合分析的经典著作,都没有像我们这样考虑排列中的路段这个课题,但有若干作者研究了交错地递增或递减的路段,因此 5 3 2 4 7 6 1 8 被认为有 4 个路段,即

5 3 2, 2 4 7, 7 6 1, 1 8

(按照 $a_1<a_2$ 或 $a_1>a_2$,头一个路段将是递增的或递减的;于是 $a_1a_2\cdots a_n$ 和 $a_n\cdots a_2a_1$ 以及$(n+1-a_1)(n+1-a_2)\cdots(n+1-a_n)$都有相同的交错路段数)。当排列 n 个元素时,这种类型路段的最大个数为 $n-1$。

试求在集合$\{1,2,\cdots,n\}$的一个随机排列中交错路段的平均数[提示:考虑(34)的证明]。

16.[*M30*] 继续上一题,设$\left\langle\!\left\langle {n \atop k} \right\rangle\!\right\rangle$是恰有 k 个交错路段的$\{1,2,\cdots,n\}$的排列数。试求一个递推关系,借助它可以计算一张$\left\langle\!\left\langle {n \atop k} \right\rangle\!\right\rangle$的表;并求生成函数 $G_n(z)=\sum_k \left\langle\!\left\langle {n \atop k} \right\rangle\!\right\rangle z^k/n!$ 的对应递推关系。试利用后一递推关系,来发现在$\{1,2,\cdots,n\}$的一个随机排列中,交错路段个数方差的一个简单公式。

17.[*M25*] 若每个 a_j 是 0 或 1,则在所有 2^n 个序列 $a_1a_2\cdots a_n$ 当中,其恰有 k 个路段者(即,出现 $k-1$ 次 $a_j>a_{j+1}$)有多少?

18.[*M28*] b_j 为范围 $0\leqslant b_j\leqslant n-j$ 中的一个整数,问:所有 $n!$个序列 $b_1b_2\cdots b_n$ 当中,(a)有多少个恰好有 k 个递降(即 $b_j>b_{j+1}$的 k 个出现)? (b)有多少个恰好有 k 个不同的元素?

▶19.[*M26*](J. Riordan) (a)在 $n\times n$ 的棋盘上可以有多少种方式,来放置 n 个不相拼(即,在同一行或同一列中没有两个)的车,使得在主对角线的给定一边内恰有 k 个? (b)在 $n\times n$ 的棋盘的主对角线之下放置 k 个不相拼的车,可以有多少种方式?

例如,图 4 表示把 8 个不相拼的车放在标准棋盘上,其中,有 3 个车放在主对角线下不带阴影的部分,这是 15619 种放置方式之一,同样也是把 3 个不相拼的车放在一个三角棋盘上的 1050 种方式之一。

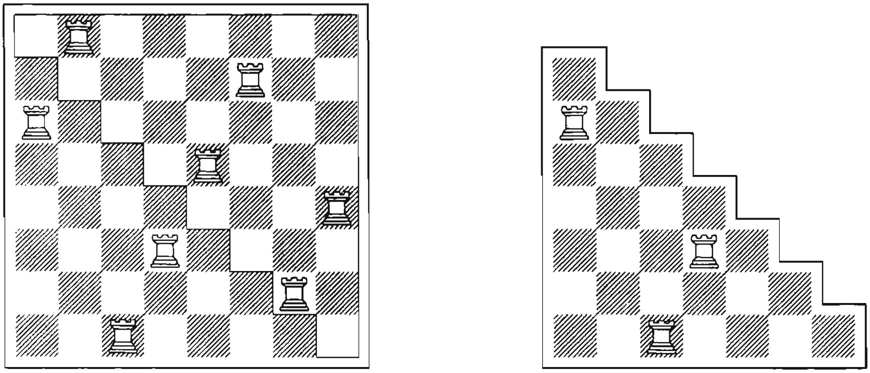


图 4　在一个棋盘上不相拼的车,且在主对角线之下有 $k=3$ 个车

▶20.[*M21*] 如果为了以非递降的次序来读完一个排列的元素,我们必须从左到右扫描它 k 次,则说这个排列是要求 k 次阅读的。例如,排列 4 9 1 8 2 5 3 6 7 要求 4 次阅读:第一次阅读时,我们得到 1,2,3;第二次,我们得到 4,5,6,7;然后 8;然后 9。试求路段和阅读之间的联系。

21.[*M22*] 如果$\{1,2,\cdots,n\}$的排列 $a_1\,a_2\cdots a_n$ 在习题 20 的意义下,有 k 个路段并要求 j 次阅读,则排列 $a_n\cdots a_2\,a_1$ 将如何?

22.[*M26*](L. Carlitz, D. P. Roselle 与 R. A. Scoville) 试证,如果 $rs<n$,则不存在具有 $n+1-r$ 个路段,且要求 s 次阅读的$\{1,2,\cdots,n\}$的排列;但如果 $n\geqslant n+1-r\geqslant s\geqslant 1$, $rs\geqslant n$,则这样的排列就存在。

23.[*HM42*](Walter Weissblum) 在排列 $a_1\,a_2\cdots a_n$ 中,在一个区段停止单调前,即在该处放置一条竖线,可得到此排列的"长路段";长路段的递增或递减取决于它们的头 2 个元素的次序,所以每个长路段的长度(可能最后一个除外)都≥2。例如,7 5|6 2|3 8 9|1 4 有 4 个长路段。试求一个无穷排列的头 2 个长路段的平均长度,并证明长路段的极限长度是

$$\left(1+\cot\frac{1}{2}\right)\Big/\left(3-\cot\frac{1}{2}\right)\approx 2.4202$$

24.[*M30*] 如同习题 5.1.1-18 那样产生的序列中,路段的平均数(作为 p 的一个函数)是多少?

25.[*M25*] 设 $U_1,\cdots,U_n$ 是在$[0,1)$中独立的一致随机数,问$\lfloor U_1+\cdots+U_n\rfloor=k$ 的概率是多少?

26.[*M20*] 设 θ 为运算 $z\,\dfrac{\mathrm{d}}{\mathrm{d}z}$,它以 n 来乘一个生成函数中 z^n 的系数。试证明:反复应用 θ 到

1/(1 - z)上 m 次的结果,可以借助于欧拉数来表达。

▶27.[*M21*] 设一个递增的森林,其节点以{1,2,…,n}来标识,并且父节点的标识小于其子节点。试证明:$\left\langle {n \atop k} \right\rangle$ 是有 $k+1$ 叶的 n 个节点递增森林的个数。

*5.1.4 图表和对合

为了完成对排列的组合性质的概述,我们将讨论把排列同整数阵列(称为图表)联系起来的某些值得注意的关系。杨氏($n_1, n_2, \cdots, n_m$)形图表(其中 $n_1 \geqslant n_2 \geqslant \cdots \geqslant n_m > 0$)是把 $n_1 + n_2 + \cdots + n_m$ 个不同的整数排成每行左对齐的一个阵列,其中第 i 行有 n_i 个元素,行中的项从左到右处于递增的次序,而且每列的项从上到下也是递增的。例如

$$\begin{array}{|c|c|c|c|c|c|} \hline 1 & 2 & 5 & 9 & 10 & 15 \\ \hline 3 & 6 & 7 & 13 \\ \cline{1-4} 4 & 8 & 12 & 14 \\ \cline{1-4} 11 \\ \cline{1-1} \end{array} \tag{1}$$

是一个杨氏(6,4,4,1)形图表。这种排法是由 Alfred Young 于 1900 年为了研究排列的矩阵表示,而引进的一个辅助手段[参见*Proc. London Math. Soc.* (2) **28** (1928), 255 ~ 292; Bruce E. Sagan, *The Symmetric Group* (Pacific Grove, Calif.: Wadsworth & Brooks/Cole, 1991)]。为了简便起见,把"杨氏图表"简称为"图表"。

一个对合是这样一种排列,它是自己的逆,例如,{1,2,3,4}的对合共有 10 个

$$\begin{array}{ccccc} \begin{pmatrix} 1&2&3&4 \\ 1&2&3&4 \end{pmatrix} & \begin{pmatrix} 1&2&3&4 \\ 2&1&3&4 \end{pmatrix} & \begin{pmatrix} 1&2&3&4 \\ 3&2&1&4 \end{pmatrix} & \begin{pmatrix} 1&2&3&4 \\ 4&2&3&1 \end{pmatrix} & \begin{pmatrix} 1&2&3&4 \\ 1&3&2&4 \end{pmatrix} \\ \begin{pmatrix} 1&2&3&4 \\ 1&4&3&2 \end{pmatrix} & \begin{pmatrix} 1&2&3&4 \\ 1&2&4&3 \end{pmatrix} & \begin{pmatrix} 1&2&3&4 \\ 2&1&4&3 \end{pmatrix} & \begin{pmatrix} 1&2&3&4 \\ 3&4&1&2 \end{pmatrix} & \begin{pmatrix} 1&2&3&4 \\ 4&3&2&1 \end{pmatrix} \end{array} \tag{2}$$

"对合"这一术语起源于古典的几何问题;这里考虑的是一般意义下的对合,这是由 H. A. Rothe 在引进逆的概念的时候,首先研究的(见 5.1.1 小节)。

我们同时讨论图表和对合可能显得有点奇怪,但是在这两个表面上无关的概念之间,实际上有着非同寻常的联系:*{1,2,…,n}的对合的个数与由元素{1,2,…,n}所能形成的图表数相同*。例如,由{1,2,3,4}恰能形成 10 张图表,即

$$\begin{array}{ccccc} \begin{array}{|c|c|c|c|} \hline 1&2&3&4 \\ \hline \end{array} & \begin{array}{|c|c|c|} \hline 1&3&4 \\ \hline 2 \\ \cline{1-1} \end{array} & \begin{array}{|c|c|} \hline 1&4 \\ \hline 2 \\ \cline{1-1} 3 \\ \cline{1-1} \end{array} & \begin{array}{|c|c|} \hline 1&3 \\ \hline 2 \\ \cline{1-1} 4 \\ \cline{1-1} \end{array} & \begin{array}{|c|c|c|} \hline 1&2&4 \\ \hline 3 \\ \cline{1-1} \end{array} \\ \\ \begin{array}{|c|c|} \hline 1&2 \\ \hline 3 \\ \cline{1-1} 4 \\ \cline{1-1} \end{array} & \begin{array}{|c|c|c|} \hline 1&2&3 \\ \hline 4 \\ \cline{1-1} \end{array} & \begin{array}{|c|c|} \hline 1&3 \\ \hline 2&4 \\ \hline \end{array} & \begin{array}{|c|c|} \hline 1&2 \\ \hline 3&4 \\ \hline \end{array} & \begin{array}{|c|} \hline 1 \\ \hline 2 \\ \hline 3 \\ \hline 4 \\ \hline \end{array} \end{array} \tag{3}$$

分别对应于 10 个对合(2)。

对合和图表之间的这一联系决不是显然的，而且大概也没有非常简单的办法来证明它。我们将要讨论的证明包含一个非常有趣的图表构造算法，它有若干其它意外的性质，该算法是以把新元素插入图表的一个特殊过程为基础的。

例如，假设要对图表

1	3	5	9	12	16
2	6	10	15		
4	13	14			
11					
17					

(4)

插入元素 8。所使用的方法是，开始把 8 放入行 1，放在原来为 9 所占据的小格上，因为 9 是在该行中大于 8 的最小元素，元素 9 就被"撞"到行 2，在那里它代替 10。然后 10 把 13 从行 3"撞"到行 4；由于行 4 没有大于 13 的元素，这一过程遂以把 13 插入到行 4 的右端而终止。至此这个图表变成

1	3	5	8	12	16
2	6	9	15		
4	10	14			
11	13				
17					

(5)

这个过程的精确描述在算法 I 中给出，同时还证明此过程始终保持图表性质。

算法 I(*插进图表*)　设 $P=(P_{ij})$是一张正整数的图表，又设 x 是不在 P 中的一个正整数。这个算法把 P 变换成另外一张图表，它除了含有 P 原来的元素外，还含有 x。这张新的图表除了在第 s 行第 t 列增加一个新的位置外，形式和旧的图表相同，这里 s 和 t 是由本算法所确定的量。

(在本算法中带圆括弧的注释用来证明它的正确性，因为容易归纳地验证这些注释是正确的，以及在整个过程中阵列 P 保持为一图表。为了方便起见，假定这图表已被镶了边，在顶上和左边有一些 0 并在右边和底下有一些∞，使得对于所有的 $i,j\geqslant 0$，P_{ij}都有定义。如果定义关系

$$a \lesssim b \quad 当且仅当\ a<b \quad 或 \quad a=b=0 \quad 或 \quad a=b=\infty \tag{6}$$

则这些图表不等式就可用下列方便的形式加以表达

$$P_{ij}=0 \quad 当且仅当\ i=0 \quad 或 \quad j=0$$
$$P_{ij}\lesssim P_{i(j+1)} \quad 和 \quad P_{ij}\lesssim P_{(i+1)j} \quad 对所有\ i,j\geqslant 0 \tag{7}$$

语句"$x\notin P$"意味着，或 $x=\infty$，或对于所有 $i,j\geqslant 0$，$x\neq P_{ij}$。)

I1.［输入 x］　置 $i\leftarrow 1$，置 $x_1\leftarrow x$，并置 j 为使得 $P_{1j}=\infty$的最小值。

I2. [求 x_{i+1}](这时 $P_{(i-1)j}<x_i<P_{ij}$ 且 $x_i \notin P$) 如果 $x_i<P_{i(j-1)}$,则 j 减 1 并重复这一步骤。否则,置 $x_{i+1}\leftarrow P_{ij}$ 且置 $r_i\leftarrow j$。

I3. [代以 x_i](现在 $P_{i(j-1)}<x_i<x_{i+1}=P_{ij}\lesssim P_{i(j+1)}$, $P_{(i-1)j}<x_i<x_{i+1}=P_{ij}\lesssim P_{(i+1)j}$,且 $r_i=j$) 置 $P_{ij}\leftarrow x_i$。

I4. [$x_{i+1}=\infty$ 吗?](现在 $P_{i(j-1)}<P_{ij}=x_i<x_{i+1}\lesssim P_{i(j+1)}$, $P_{(i-1)j}<P_{ij}=x_i<x_{i+1}\lesssim P_{(i+1)j}$, $r_i=j$,且 $x_{i+1}\notin P$) 若 $x_{i+1}\neq\infty$, i 增 1,并返回步骤 I2。

I5. [确定 s,t]置 $s\leftarrow i$, $t\leftarrow j$,并终止此算法。此时,满足条件

$$P_{st}\neq\infty, P_{(s+1)t}=P_{s(t+1)}=\infty \tag{8}$$

▌

算法 I 定义了一个"碰撞序列"

$$x=x_1<x_2<\cdots<x_s<x_{s+1}=\infty \tag{9}$$

和一个辅助的列标序列

$$r_1\geqslant r_2\geqslant\cdots\geqslant r_s=t \tag{10}$$

对于 $1\leqslant i\leqslant s$,元素 P_{ir_i} 已经从 x_{i+1} 变成为 x_i。例如,当把 8 插到(4)中时,碰撞序列是 8,9,10,13,∞,且辅助序列为 4,3,2,2。我们可以用另一种形式表述这个算法,以便它使用更少的临时存储;只有 j, x_i 和 x_{i+1} 的当前值需要记住。但这里引入了序列(9)和(10),为的是能够证明关于这个算法的一些有趣的事情。

关于算法 I,将要使用的键码事实在于它能够被"逆转运行":给定在步骤 I5 中确定的 s 和 t 的值,能够倒过来把 P 转换成为它原来的形式,并确定和消去被插入的元素 x。例如,考虑(5)并假设告诉我们元素 13 是在原为空格的位置上,则 13 必然是被 10 从行 3 撞下的,因为 10 是在该行中小于 13 的最大元素;类似地,10 必然是被 9 从行 2 撞下的,而 9 必然是被 8 从行 1 撞下的。于是,可以从(5)倒回到(4)。下列算法详细地说明这一过程。

算法 D(从一图表删去) 给定一个图表 P 以及满足(8)的正整数 s,t,本算法把 P 变换成另一个具有几乎相同形状的图表,但在第 s 行第 t 列处具有∞。由本算法所确定的一个元素 x 被从 P 删去。

(如同在算法 I 中一样,这里加上了带圆括弧的断言,以便于证明在整个过程中 P 保持为图表。)

D1. [输入 s,t] 置 $j\leftarrow t$, $i\leftarrow s$, $x_{s+1}\leftarrow\infty$。

D2. [求 x_i] (这时 $P_{ij}<x_{i+1}\lesssim P_{(i+1)j}$ 且 $x_{i+1}\notin P$) 如果 $P_{i(j+1)}<x_{i+1}$,则 j 增加 1 并重复这一步骤。否则,置 $x_i\leftarrow P_{ij}$ 且 $r_i\leftarrow j$。

D3. [代以 x_{i+1}] (现在 $P_{i(j-1)}<P_{ij}=x_i<x_{i+1}\lesssim P_{i(j+1)}$, $P_{(i-1)j}<P_{ij}=x_i<x_{i+1}\lesssim P_{(i+1)j}$,且 $r_i=j$) 置 $P_{ij}\leftarrow x_{i+1}$。

D4. [$i=1$ 吗?] (现在 $P_{i(j-1)}<x_i<x_{i+1}=P_{ij}\lesssim P_{i(j+1)}$, $P_{(i-1)j}<x_i<x_{i+1}=P_{ij}\lesssim P_{(i+1)j}$,且 $r_i=j$) 如果 $i>1$,则 i 减 1 并返回步骤 D2。

D5.［确定 x］　置 $x \leftarrow x_1$ 并终止这个算法(现在 $0<x<\infty$)。　▍

在算法 I 和算法 D 中出现的带圆括弧的断言不仅是证明这些算法保持图表结构的有用方式，也有利于验证*算法 I 和 D 恰为互逆*。给定某个图表 P 和某个正整数 $x \notin P$，如果首先实施算法 I，它将插入 x，并确定满足(8)的正整数 s,t；对此结果应用算法 D，则将重新算出 x 并将恢复 P。反过来，给定某个图表 P 和某个满足(8)的整数 s,t，如果首先实施算法 D，它将修改 P，删去某个正整数 x；对此结果应用算法 I，将重新算出 s、t 并将恢复 P。原因是步骤 I3 和 D4 的带圆括弧的断言是相同的，步骤 I4 和 D3 的断言也一样，而且这些断言惟一地表征 j 的值。因此辅助序列(9)，(10)在每种情况下都是一样的。

现在已做好准备来证明图表的一个基本性质。

定理 A　*在 $\{1,2,\cdots,n\}$ 的所有排列的集合，与由 $\{1,2,\cdots,n\}$ 形成的图表的有序对偶 (P,Q) 的集合之间，有一对一的对应，其中 P 和 Q 有相同的形状。*

(这个定理的一个例子出现于下边的证明内。)

证明　证明一个稍微更一般的结果是方便的。给定任何两行的阵列

$$\begin{pmatrix} q_1 & q_2 & \cdots & q_n \\ p_1 & p_2 & \cdots & p_n \end{pmatrix}, \qquad \begin{array}{l} q_1 < q_2 < \cdots < q_n \\ p_1, p_2, \cdots, p_n \text{ 不同} \end{array} \tag{11}$$

我们将构造两个对应的图表 P 和 Q，其中 P 的元素是 $\{p_1,\cdots,p_n\}$，而 Q 的元素是 $\{q_1,\cdots,q_n\}$，并且 P 的形状即是 Q 的形状。

设 P 和 Q 开始是空的。然后，对于 $i=1,2,\cdots,n$(以这个次序)做下列操作：利用算法 I 把 p_i 插入图表 P；然后置 $Q_{st} \leftarrow q_i$，其中 s 和 t 确定 P 中的新被填入的位置。

例如，如果给定的排列是 $\begin{pmatrix} 1 & 3 & 5 & 6 & 8 \\ 7 & 2 & 9 & 5 & 3 \end{pmatrix}$，我们得到

	P	Q
插入 7：	$\begin{array}{\|c\|}\hline 7 \\ \hline\end{array}$	$\begin{array}{\|c\|}\hline 1 \\ \hline\end{array}$
插入 2：	$\begin{array}{\|c\|}\hline 2 \\ \hline 7 \\ \hline\end{array}$	$\begin{array}{\|c\|}\hline 1 \\ \hline 3 \\ \hline\end{array}$
插入 9：	$\begin{array}{\|c\|c\|}\hline 2 & 9 \\ \hline 7 \\ \cline{1-1}\end{array}$	$\begin{array}{\|c\|c\|}\hline 1 & 5 \\ \hline 3 \\ \cline{1-1}\end{array}$
插入 5：	$\begin{array}{\|c\|c\|}\hline 2 & 5 \\ \hline 7 & 9 \\ \hline\end{array}$	$\begin{array}{\|c\|c\|}\hline 1 & 5 \\ \hline 3 & 6 \\ \hline\end{array}$
插入 3：	$\begin{array}{\|c\|c\|}\hline 2 & 3 \\ \hline 5 & 9 \\ \hline 7 \\ \cline{1-1}\end{array}$	$\begin{array}{\|c\|c\|}\hline 1 & 5 \\ \hline 3 & 6 \\ \hline 8 \\ \cline{1-1}\end{array}$

(12)

于是对应于 $\begin{pmatrix} 1 & 3 & 5 & 6 & 8 \\ 7 & 2 & 9 & 5 & 3 \end{pmatrix}$ 的图表 (P,Q) 是

$$P = \begin{array}{|c|c|}\hline 2 & 3 \\ \hline 5 & 9 \\ \hline 7 & \\ \cline{1-1}\end{array}\,, \qquad Q = \begin{array}{|c|c|}\hline 1 & 5 \\ \hline 3 & 6 \\ \hline 8 & \\ \cline{1-1}\end{array} \tag{13}$$

由这种构造法，显然 P 和 Q 总有相同的形状；其次，由于总是以递增的次序在 Q 的周围增加元素，故 Q 是一个图表。

反之，给定两个相同形状的图表 P 和 Q，我们可以求相应的两行阵列(11)如下。设 Q 的元素是

$$q_1 < q_2 < \cdots < q_n$$

对于 $i = n, \cdots, 2, 1$(以这个次序)，令 p_i 是当应用算法 D 于 P，并且用使得 $Q_{st} = q_i$ 的值 s、t 时，被消去的元素 x。

例如，这个构造将以(13)开始，并将逐次地做(12)的反运算直到 P 为空，这就得到 $\begin{pmatrix}1 & 3 & 5 & 6 & 8\\ 7 & 2 & 9 & 5 & 3\end{pmatrix}$。

由于算法 I 和 D 彼此互逆，故已经描述的两个构造彼此互逆，而且建立了一一对应。 ▌

在定理 A 的证明中，定义的对应关系有许多惊人的性质，现在就着手来推导其中的某些。读者不妨亲手试试习题 1 中的例子，以便在往下阅读之前能熟悉这个构造。

一旦已经从行 1 把一个元素撞到行 2 中，它就不再影响行 1 了；而且由“被撞”元素序列来构造行 2，3，… 的方法，恰恰就像由原来的排列来构造行 1，2，… 一样。这些事实提示我们，可以用另一种方式来考察定理 A 的构造，即仅专注于 P 和 Q 的开头几行。例如，按照(12)，排列 $\begin{pmatrix}1 & 3 & 5 & 6 & 8\\ 7 & 2 & 9 & 5 & 3\end{pmatrix}$ 引起在行 1 中发生下列动作：

$$\begin{array}{l}
1\text{:插入 }7\text{，置 }Q_{11}\leftarrow 1\text{。}\\
3\text{:插入 }2\text{，撞下 }7\text{。}\\
5\text{:插入 }9\text{，置 }Q_{12}\leftarrow 5\text{。}\\
6\text{:插入 }5\text{，撞下 }9\text{。}\\
8\text{:插入 }3\text{，撞下 }5\text{。}
\end{array} \tag{14}$$

于是 P 的头一行是 2 3，而 Q 的头一行是 1 5，其次，P 和 Q 中剩下的行是对应于“被撞下”的两行阵列

$$\begin{pmatrix}3 & 6 & 8\\ 7 & 9 & 5\end{pmatrix} \tag{15}$$

的图表。为了研究行 1 上构造的特性，可以考察进入该行的一个给定列的诸元素。我们说，(q_i, p_i) 相对于两行阵列

$$\begin{pmatrix}q_1 & q_2\cdots q_n\\ p_1 & p_2\cdots p_n\end{pmatrix}, \quad \begin{array}{l} q_1 < q_2 < \cdots < q_n \\ p_1, p_2, \cdots, p_n \text{ 不同}\end{array} \tag{16}$$

来说是在 类 t 中,如果从一个空表格 P 开始,在逐次应用算法 I 于 $p_1,p_2,\cdots,p_i$ 以后,$p_i=P_{1t}$的话(记住,算法 I 总是把给定的元素插入到行 1 中)。

容易看出,当且仅当 p_i 有 $i-1$ 个反序,即 $p_i=\min\{p_1,p_2,\cdots,p_i\}$是"自左到右的极小值"时,$(q_ip_i)$在类 1 中。如果删去(16)中属于类 1 的诸列,则得到另一个两行阵列

$$\begin{pmatrix} q'_1 & q'_2\cdots q'_m \\ p'_1 & p'_2\cdots p'_m \end{pmatrix} \tag{17}$$

使得(q,p)相对于(17)来说是在类 t 中,当且仅当它相对于(16)来说是在类 $t+1$ 中。从(16)进行到(17)的操作表示撤消行 1 最左的位置。这给了一个确定这些类的系统方法。例如,在$\begin{pmatrix}1&3&5&6&8\\7&2&9&5&3\end{pmatrix}$中,自左到右极小的元素是 7 和 2,所以类 1 是$\{(1,7),(3,2)\}$;在剩下的阵列$\begin{pmatrix}5&6&8\\9&5&3\end{pmatrix}$中,所有的元素都是极小的,所以类 2 是$\{(5,9),(6,5),(8,3)\}$。在"被撞下"的阵列(15)中,类 1 是$\{(3,7),(8,5)\}$,而类 2 是$\{(6,9)\}$。

对于任何固定的 t 值,类 t 的元素可以标记成

$$(q_{i_1},p_{i_1}),\cdots,(q_{i_k},p_{i_k})$$

其中

$$\begin{gathered} q_{i_1}<q_{i_2}<\cdots<q_{i_k} \\ p_{i_1}>p_{i_2}>\cdots>p_{i_k} \end{gathered} \tag{18}$$

因为随着插入算法的进行,图表位置 P_{1t}取的值是递减序列$p_{i_1},\cdots,p_{i_k}$。在构造结束时,有

$$P_{1t}=p_{i_k},Q_{1t}=q_{i_1} \tag{19}$$

P 和 Q 的第 2,3,…行是通过"被撞下来的"两行阵列定义的,这个阵列包含

$$\begin{pmatrix} q_{i_2} & q_{i_3} & \cdots & q_{i_k} \\ p_{i_1} & p_{i_2} & \cdots & p_{i_{k-1}} \end{pmatrix} \tag{20}$$

诸列,加上以类似方式从其它类形成的其它列。

这些发现导致了通过手算来算出 P 和 Q 的一个简单方法(见习题 3),而且它们也为我们提供了证明一个颇为意外的结果的手段。

定理 B *如果排列*

$$\begin{pmatrix} 1 & 2 & \cdots & n \\ a_1 & a_2 & \cdots & a_n \end{pmatrix}$$

对应于定理 A 的构造中的图表(P,Q),则逆排列对应于(Q,P)。

这项事实是十分惊人的,因为 P 和 Q 在定理 A 中是通过完全不同的方法形成的,而且一个排列的逆,是通过稍微反复无常地捣动两行阵列的诸列而得到的。

证明 假设有一个两行阵列(16),交换这些行,并重排诸列,使得新的顶上一行以递增次序出现,就得到"逆"阵列

$$\begin{pmatrix} q_1 & q_2 & \cdots & q_n \\ p_1 & p_2 & \cdots & p_n \end{pmatrix}^- = \begin{pmatrix} p_1 & p_2 & \cdots & p_n \\ q_1 & q_2 & \cdots & q_n \end{pmatrix} = \begin{pmatrix} p'_1 & p'_2 & \cdots & p'_n \\ q'_1 & q'_2 & \cdots & q'_n \end{pmatrix}, \quad \begin{matrix} p'_1 < p'_2 < \cdots < p'_n \\ q'_1, q'_2, \cdots, q'_n \text{不同} \end{matrix} \tag{21}$$

我们将证明这项操作对应于在定理 A 的构造中交换 P 和 Q。

习题 2 已重新叙述了如何确定类的注释,使得(q_i,p_i)的类不依赖于 $q_1, q_2, \cdots, q_n$ 是以递增次序出现的这一事实。由于得到的条件在诸 q 和诸 p 中是对称的,故操作(21)不破坏类的结构;如果(q,p)相对于(16)来说是处于类 t 中,则(p,q)相对于(21)来说就处于类 t 中。如果因此通过同(18)的类似性我们把后一类 t 的元素排成

$$\begin{matrix} p_{i_k} < \cdots < p_{i_2} < p_{i_1} \\ q_{i_k} > \cdots > q_{i_2} > q_{i_1} \end{matrix} \tag{22}$$

则如同在(19)中一样,就有

$$P_{1t} = q_{i_1}, \quad Q_{1t} = p_{i_k} \tag{23}$$

而且诸列

$$\begin{pmatrix} p_{i_{k-1}} & \cdots & p_{i_2} & p_{i_1} \\ q_{i_k} & \cdots & q_{i_3} & q_{i_2} \end{pmatrix} \tag{24}$$

如同在(20)中一样进入到"撞下的"阵列中。因此 P 和 Q 的前几行被交换。而且(21)的"撞下的"两行阵列是(16)的"撞下的"两行阵列的逆,所以,通过对图表中的行数用归纳法,就完成了这一证明。 ▎

推论 *由$\{1,2,\cdots,n\}$所能形成的图表的数目是$\{1,2,\cdots,n\}$上的对合的数目。*

证明 如果 π 是对应于(P,Q)的一个对合,则 $\pi=\pi^-$ 对应于(Q,P);因此 $P=Q$。反之,如果 π 是对应于(P,P)的任何排列,则 π^- 也对应于(P,P),因此 $\pi=\pi^-$。所以,在对合 π 和图表 P 之间存在一一对应。 ▎

显然,一个图表在左上角的元素总是最小的。这提示了对一个数集进行排序的一种可能的途径:首先,可以通过反复地使用算法 I,把它们放置到一个图表当中;这就把最小的元素放到角上。然后删去这个最小的元素,并重新安排剩余的元素,使得它们形成另一个图表;然后删去新的最小元素;等等。

因此,让我们来考虑当从图表

1	3	5	7	11	15
2	6	8	14		
4	9	13			
10	12				
16					

(25)

删去角元素时，将出现什么情况。如果删去 1，则 2 必须取代它的位置。然后可以向上移动 4 到 2 原来所在的位置，但不能把 10 移动到 4 原来所在的位置；而是移动 9，然后用 12 代替 9。如此下去，就得到如下的过程：

算法 S(删去角元素)　给定一个图表 P，本算法删去 P 的左上角元素，并且移动其他的元素，使得 P 仍保持图表的性质。这里沿用算法 I 和 D 的记号约定。

S1.［初始化］　置 $r \leftarrow 1, s \leftarrow 1$。

S2.［完成?］　如果 $P_{rs} = \infty$，则此过程完成。

S3.［比较］　如果 $P_{(r+1)s} \lesssim P_{r(s+1)}$，则转到步骤 S5(考察恰在空格下方和右方的元素，并移动其中较小者)。

S4.［左移］　置 $P_{rs} \leftarrow P_{r(s+1)}, s \leftarrow s+1$，并返回 S3。

S5.［上移］　置 $P_{rs} \leftarrow P_{(r+1)s}, r \leftarrow r+1$，并返回 S2。　▌

容易证明，在算法 S 已经删去了 P 的角元素之后，P 仍然是一个图表(见习题 10)。所以如果重复算法 S 直到 P 为空，则就能以递增的次序读出它的元素。可惜，这并不是像将看到的其它方法一样有效的排序算法；它的极小运行时间与 $n^{1.5}$ 成比例，而使用树形而不是使用图表结构的类似算法，却有阶为 $n\log_2 n$ 的执行时间。

尽管算法 S 不能导致一个特别有效的排序算法，但它却有某些有趣的性质。

定理 C(M. P. Schützenberger)　*如果 P 是由排列 $a_1\ a_2 \cdots a_n$ 通过定理 A 的构造形成的图表，而且如果 $a_i = \min\{a_1, a_2, \cdots, a_n\}$，则算法 S 把 P 变成为对应于 $a_1 \cdots a_{i-1} a_{i+1} \cdots a_n$ 的图表。*

证明　见习题 13。　▌

在应用算法 S 于一图表之后，把删去的元素放置到刚刚空出来的位置 P_{rs} 中，并用斜体来指出它实际上不是这个图表的一部分。例如在应用这一过程于图表(25)之后，我们将有

2	3	5	7	11	15
4	6	8	14		
9	12	13			
10	*1*				
16					

再应用两次，得出

4	5	7	11	15	*2*
6	8	13	14		
9	12	*3*			
10	*1*				
16					

继续进行,直到所有元素都已被删去为止,给出

16	14	13	12	10	2
15	9	6	4		
11	5	3			
8	1				
7					

(26)

它同原来的图表(25)有同样的形状。这个结构可以称做一个对偶图表,因为它除了使用“对偶次序”(逆转<和>的作用)这一点外,其它都像一个图表。我们以符号 P^S 表示以这种方式由 P 形成的对偶图表。

由 P^S 可以惟一地确定 P;事实上,通过应用完全相同的算法(除了逆转斜体和正体的次序和角色外),可以从 P^S 得到原来的图表 P,因为 P^S 是一个对偶图表。例如,应用这个算法于(26)两步,将给出

14	13	12	10	2	15
11	9	6	4		
8	5	3			
7	1				
16					

并最终将重新产生出(25)来!这个值得注意的事实是下一定理的推论之一。

定理 D(C. Schensted, M. P. Schützenberger) *设*

$$\begin{pmatrix} q_1 & q_2 & \cdots & q_n \\ p_1 & p_2 & \cdots & p_n \end{pmatrix} \tag{27}$$

是对应于图表(P,Q)的两行阵列。

a)*对诸 q,而不是对诸 p 应用对偶(逆转)次序,则两行阵列*

$$\begin{pmatrix} q_n & \cdots & q_2 & q_1 \\ p_n & \cdots & p_2 & p_1 \end{pmatrix} \tag{28}$$

对应于(P^T,$(Q^S)^T$)。

像通常一样,“T”表示对行和列进行转置的操作;P^T 是一个图表,而$(Q^S)^T$ 是一个对偶图表,因为诸 q 的次序被逆转了。

b)*对诸 p 而不是对诸 q 应用对偶次序,则两行阵列*(27)*对应于*($(P^S)^T$,Q^T)。

c)*对诸 p 和诸 q 同时应用对偶次序,则两行阵列*(28)*对应于*(P^S,Q^S)。

证明 这个定理的简单证明尚不可知。对于某个对偶图表 X,情况 a)对应于(P^T,X)这一事实,在习题 5 中证明;因此由定理 B,对于某个对偶图表 Y,情况 b)对应于(Y,Q^T),且必然有 P^T 的形状。

设 $p_i=\min\{p_1,\cdots,p_n\}$;由于 p_i 是对偶次序下“最大的”元素,它出现于 Y 的外围上,而且它不撞下定理 A 的构造中的任何元素。于是,如果利用对偶次序,逐次地

插入 $p_1,\cdots,p_{i-1},p_{i+1},\cdots,p_n$，则就得到 $Y-\{p_i\}$，即除去了 p_i 的 Y。由定理 C，如果利用正常的次序，逐次地插入 $p_1,\cdots,p_{i-1},p_{i+1},\cdots,p_n$，则得到对 p 使用算法 S 而得到的图表 $d(P)$。对 n 用归纳法，$Y-\{p_i\}=(d(P)^S)^T$。但由于按操作 S 的定义

$$(P^S)^T-\{p_i\}=(d(P)^S)^T \tag{29}$$

以及由于 Y 有和 $(P^S)^T$ 同样的形状，必然有 $Y=(P^S)^T$。

这就证明了 b)，而且通过应用定理 B 而得出 a)。相继地应用 a) 和 b)，可证明情况 c) 对应于 $(((P^T)^S)^T,((Q^S)^T)^T)$；而这就是 (P^S,Q^S)，因为由操作 S 的行、列对称性可知 $(P^S)^T=(P^T)^S$。 ▌

特别是，这个定理证实了关于图表插入算法的两个惊人的事实：如果逐次插入不同的元素 $p_1,\cdots,p_n$ 到一个空图表中可产生图表 P，则按相反的次序插入 $p_n,\cdots,p_1$ 就产生转置图表 P^T。而如果不仅以这个次序 $p_n,\cdots,p_1$ 插入诸 p，并且在插入过程中也交换 $<$ 和 $>$ 以及 0 和 ∞ 的作用，则就得到对偶图表 P^S。读者不妨用一些简单的例子来试验这些过程。这些巧合的不寻常的属性可能使我们猜测：在舞台后边有某些魔法在操纵！还不知道关于这些现象的简单的解释；甚至没有显然的方式来证明下列事实：情况 c) 对应于与 P 和 Q 有相同形状的图表，尽管习题 2 中的类的特征确实提供了一个重要的提示。

G. de B. Robinson[*American J. Math.* **60**(1938)，745～760，§5]以稍微含混和不同的形式，给出了定理 A 中的对应关系，作为解决群论中颇为困难的问题的一部分。他未加证明地叙述了定理 B。许多年后，C. Schensted 独立地重新发现了这个对应关系，他所指出的这一对应关系，实质上是借助于我们在算法 I 中所做的那样的"碰撞"给出的，Schensted 还说明了定理 D 的 a) 的"P"部分[*Canadian J. Math.* **13**(1961)，179～191]。M. P. Schützenberger[*Math. Scand.* **12**(1963)，117～128]证明了定理 C 和定理 D(a) 的"Q"部分，由它们就得出了 b) 和 c)。有可能把这个对应关系推广到多重集合的排列；Schensted 考虑了 $p_1,\cdots,p_n$ 不必是不同的情况，而 Knuth 研究了"最终"推广到诸 p 和诸 q 两者皆可以包含重复元素的情况[*Pacific J. Math.* **34** (1970)，709～727]。

现在我们转到一个有关的问题：*有多少个由 $\{1,2,\cdots,n\}$ 形成的图表有一个给定的形状 $(n_1,n_2,\cdots,n_m)$，其中 $n_1+n_2+\cdots+n_m=n$？* 如果用 $f(n_1,n_2,\cdots,n_m)$ 来表示这个数，而且允许参数 n_j 为任意整数，则函数 f 必然满足下列关系：

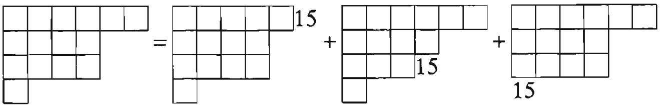
$$f(n_1,n_2,\cdots,n_m)=0 \quad 除非 \quad n_1\geqslant n_2\geqslant\cdots\geqslant n_m\geqslant 0 \tag{30}$$

$$f(n_1,n_2,\cdots,n_m,0)=f(n_1,n_2,\cdots,n_m) \tag{31}$$

$$f(n_1,n_2,\cdots,n_m)=f(n_1-1,n_2,\cdots,n_m)+f(n_1,n_2-1,\cdots,n_m)+\cdots$$
$$+f(n_1,n_2,\cdots,n_m-1) \quad 如果\ n_1\geqslant n_2\geqslant\cdots\geqslant n_m\geqslant 1 \tag{32}$$

递推式(32)得自这样一个事实，即撤消一个图表的最大元素后总是得到另一个图表；例如，形状为(6,4,4,1)的图表个数是 $f(5,4,4,1)+f(6,3,4,1)+f(6,4,3,1)$

$+f(6,4,4,0)=f(5,4,4,1)+f(6,4,3,1)+f(6,4,4)$，因为$\{1,2,\cdots,15\}$上形状为$(6,4,4,1)$的每个图表，均由插入元素15到形状为$(5,4,4,1)$，$(6,4,3,1)$或$(6,4,4)$的一个图表的适当位置而形成。如下图



(33)

满足这些关系的函数$f(n_1,n_2,\cdots,n_m)$有一个相当简单的形式

$$f(n_1,n_2,\cdots,n_m)=\frac{\Delta(n_1+m-1,n_2+m-2,\cdots,n_m)n!}{(n_1+m-1)!(n_2+m-2)!\cdots n_m!} \tag{34}$$

假定相当温和的条件

$$n_1+m-1\geqslant n_2+m-2\geqslant\cdots\geqslant n_m$$

满足；这里Δ表示下列"判别式的平方根"函数

$$\Delta(x_1,x_2,\cdots,x_m)=\det\begin{pmatrix} x_1^{m-1} & x_2^{m-1} & \cdots & x_m^{m-1} \\ \vdots & \vdots & & \vdots \\ x_1^2 & x_2^2 & & x_m^2 \\ x_1 & x_2 & & x_m \\ 1 & 1 & \cdots & 1 \end{pmatrix}=\prod_{1\leqslant i<j\leqslant m}(x_i-x_j) \tag{35}$$

公式(34)是G. Frobenius在解决群论中的一个等价问题时导出的[*Sitzungsberichte Preuβ. Akad. der Wissenschaften* (1900)，516～534，§3]，并且利用了一个颇为深入的群论上的论证。MacMahon独立地给出了一个组合的证明[*Philosophical Trans.* **A209**(1909)，153～175]。这个公式可以通过归纳法来建立，因为关系(30)和(31)是容易证明的，而且在习题17的恒等式中置$y=-1$就得出(32)。

定理A给出了与图表数的这个公式有关的一个值得注意的恒等式。如果对所有形状求和，则有

$$n!=\sum_{\substack{k_1\geqslant k_2\geqslant\cdots\geqslant k_n\geqslant 0\\ k_1+k_2+\cdots+k_n=n}} f(k_1,k_2,\cdots,k_n)^2=$$

$$n!^2\sum_{\substack{k_1\geqslant k_2\geqslant\cdots\geqslant k_n\geqslant 0\\ k_1+k_2+\cdots+k_n=n}}\frac{\Delta(k_1+n-1,k_2+n-2,\cdots,k_n)^2}{(k_1+n-1)!^2(k_2+n-2)!^2\cdots k_n!^2}=$$

$$n!^2\sum_{\substack{q_1>q_2>\cdots>q_n\geqslant 0\\ q_1+q_2+\cdots+q_n=(n+1)n/2}}\frac{\Delta(q_1,q_2,\cdots,q_n)^2}{q_1!^2q_2!^2\cdots q_n!^2}$$

因此

$$\sum_{\substack{q_1+q_2+\cdots+q_n=(n+1)n/2\\ q_1,q_2,\cdots,q_n\geqslant 0}}\frac{\Delta(q_1,q_2,\cdots,q_n)^2}{q_1!^2q_2!^2\cdots q_n!^2}=1 \tag{36}$$

在后面这个和式中，已经除去了不等式$q_1>q_2>\cdots>q_n$，因为被加数是诸q的对称

函数,当 $q_i=q_j$ 时它为 0。一个类似的恒等式在习题 24 中出现。

基于“钩子”的思想,可以用一种更有趣的方式来表达求图表个数的公式。对应于图表中的每个小格,可定义钩子为小格本身加上位于其下边和右边的诸小格。例如,图 5 的阴影区域是对应于行 2、列 3 的小格(2,3)的钩子;它包含 6 个小格。图 5 的每个小格已经填以它的钩子的长度。

12	11	8	7	5	4	1
10	9	6	5	3	2	●
9	8	5	4	2	1	●
6	5	2	1	●		
3	2	●				
2	1	●				

图 5　钩子及钩子的长度

如果图表的形状为$(n_1,n_2,\cdots,n_m)$,则最长钩子的长度为 n_1+m-1。对钩子长度作进一步的考察表明,行 1 包含所有长度 $n_1+m-1,n_1+m-2,\cdots,1$,但不包括$(n_1+m-1)-(n_m)$, $(n_1+m-1)-(n_{m-1}+1),\cdots,(n_1+m-1)-(n_2+m-2)$。例如,在图 5 中,行 1 的钩子长度是 12,11,10,…,1 中除 10,9,6,3,2 之外的数;这些例外对应于 5 个不存在的钩子,即从不存在的诸小格(6,3),(5,3),(4,5),(3,7),(2,7)通到小格(1,7)的钩子。类似地,行 j 包含有所有的钩子长度 $n_j+m-j,\cdots,1$,但不包括$(n_j+m-j)-(n_m),\cdots,(n_j+m-j)-(n_{j+1}+m-j-1)$。由此得出所有钩子长度的乘积等于

$$\frac{(n_1+m-1)!(n_2+m-2)!\cdots n_m!}{\Delta(n_1+m-1,n_2+m-2,\cdots n_m)}$$

这恰是等式(34)中的内容,所以我们就导出了由 J.S.Frame,G. de B.Robinson 以及 R.M.Thrall[*Canadian J. Math.* **6** (1954),316～318]所得出的受人赞誉的结果:

定理 H　*$\{1,2,\cdots,n\}$上的有一个确定形状的图表数是 n! 除以钩子长度的乘积。*

既然是如此简单的规则,它应该有一个简单的证明。一个启发性的论证进行:这个图表的每个元素在它的钩子中是最小的。如果随机地填图表,则小格(i,j)包含对应钩子的极小元素的概率是钩子长度的倒数。对于所有的 i 和 j 求这些概率的乘积即给出定理 H。但不幸地是,这个论证是荒谬的,因为这些概率远不是独立的。尽管研究者们确实发现了很多有启发性的间接证明(习题 35,36 和 38),但直到 1992 年以前(见习题 39),基于正确使用钩子组合性质的定理 H 尚未有直接的证明。

定理 H 同我们在第 2 章中所考虑的树的枚举有着有趣的联系。我们注意到,有 n 个节点的二叉树对应于可通过一个找得到的排列,而且这样的排列对应于 n 个 S 和 n 个 X 的那些序列 $a_1a_2\cdots a_{2n}$,其中当自左向右阅读时,S 的个数决不少于 X 的个数(见习题 2.2.1-3 和 2.3.1-6)。后面的那些序列以自然的方式对应于形状为(n,n)的图表;我们在行 1 放置使得 $a_i=\mathrm{S}$ 的下标 i,而在行 2 中放置使得 $a_i=\mathrm{X}$ 的下标 i。例如,序列

S　S　S　X　X　S　S　X　X　S　X　X

对应于图表

1	2	3	6	7	10
4	5	8	9	11	12

(37)

在这个图表中,当且仅当自左向右读 X 的个数决不超过 S 的个数时,关于列的限制成立。由定理 H,形如(n,n)的图表的数目是

$$\frac{(2n)!}{(n+1)!\,n!}$$

而这是二叉树的数目(同等式 2.3.4.4-(14)一致)。而且,如果对于 $n\geqslant m$ 使用形状为(n,m)的图表,那么这个论证解决了在习题 2.2.1-4 的答案中考虑的,更为一般的“抽签”问题。所以,定理 H 包括了某些颇为复杂的枚举问题作为简单的特殊情况。

元素$\{1,2,\cdots,2n\}$上形状为(n,n)的任何图表 A 按 MacMahon[*Combinatory Analysis* **1** (1915),130~131]提出的如下方式对应于相同形状的两个图表(P,Q)。设 P 由在A 中出现的元素$\{1,\cdots,n\}$组成;Q 由剩下的元素转动 180°,并分别以 n,$n-1,\cdots,1$ 代替$n+1,n+2,\cdots,2n$ 形成。例如,(37)分成为

1	2	3	6
4	5		

和

		7	10
8	9	11	12

对后者进行转动和更名就得到

$P=$

1	2	3	6
4	5		

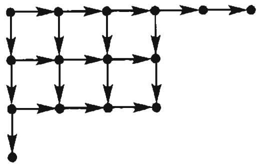
$Q=$

1	2	4	5
3	6		

(38)

反之,任何一对至多两行的各有 n 个小格相等形状的图表,都以这样一种方式对应于形状为(n,n)的图表。因此由习题 7,*不包含递减子序列* $a_i>a_j>a_k(i<j<k)$*的* $\{1,2,\cdots,n\}$*的排列* $a_1a_2\cdots a_n$ *的个数,为具有* n *个节点的二叉树的个数*。D. Rotem [*Combinatory Analysis* **1**(1915),130~131]发现了在这样的排列和二叉树之间有一个有趣的一一对应,它较之这里已经使用的经由算法 I 的迂回(兜圈)方法更直接。类似地,在二叉树和对于 $i<j<k$ 没有 $a_j>a_k>a_i$ 情形的排列之间,有着颇为直接的对应(见习题 2.2.1-5)。

填写一个(6,4,4,1)形的图表的方式数,显然是把标号$\{1,2,\cdots,15\}$,以如下方式放置到有向图的顶点的方式数



(39)

这种放置的方式就是每当 $u\to v$ 时,顶点 u 的标号小于顶点 v 的标号。换言之,它是在 2.2.3 小节的意义下,对偏序(39)进行拓扑排序的方式数。

一般说来,我们可以对不包含有向回路的任何有向图问同样的问题。如果有把定理 H 推广到任意有向图的某个简单的公式,那就好了;但并非所有的图都像对应于图表的图一样有这样令人高兴的性质。本节末尾的习题中讨论了某些其它类的有向图,对于它们来说,标号问题有一个简单的解。也还有一些习题,说明某些有向图没有对应于定理 H 的简单公式。例如,加标号的方式数不总是 $n!$ 的一个因子。

为了完成我们的研究,我们来计算可以由 n 个不同的元素形成的图表总数;并以 t_n 来表示这个数。由定理 B 的推论,t_n 是$\{1,2,\cdots,n\}$的对合数。当且仅当一个排列的轮换形式仅由一元轮换(不动点)和二元轮换(对换)所组成时,这个排列是它自己的逆。由于 t_n 个对合中的t_{n-1}个有一元轮换(n),且它们中的 t_{n-2}个有二元轮换(jn)(对于固定的$j<n$),得到公式

$$t_n = t_{n-1} + (n-1)t_{n-2} \tag{40}$$

这是 Rothe 于 1800 年设计出来的,以便对于小的 n 造出 t_n 表。对于 $n \geqslant 0$ 的值是 1,1,2,4,10,26,76,232,2620,9496,…。

用另一种方式来计数时,假设有 k 个二元轮换和$(n-2k)$个一元轮换。有$\binom{n}{2k}$种方法来选择不动点,且多项式系数$(2k)!/(2!)^k$ 是把其它元素排成k 个不同的转置的方式数;除以 $k!$ 使得转置是不加区别的,因此得到

$$t_n = \sum_{k=0}^{\lfloor n/2 \rfloor} t_n(k), \quad t_n(k) = \frac{n!}{(n-2k)!2^k k!} \tag{41}$$

可惜,这个和数没有简单的封闭形式(除非我们选择把 Hermite 多项式 $i^n 2^{-n/2} Hn(-i/\sqrt{2})$当做简单的多项式),所以可用两个间接的方法以得到对于 t_n 的更好的理解:

a)可以找到生成函数

$$\sum_n t_n z^n / n! = e^{z+z^2/2} \tag{42}$$

见习题 25。

b)可以确定 t_n 的渐近特性。这是一个非常有启发性的问题,因为它包括某些在其它方面对我们有用的一般性技术,所以我们将以分析 t_n 的渐近特性来结束这一节。

分析(41)渐近特性的头一步,是判明它对于和数的主要贡献。由于

$$\frac{t_n(k+1)}{t_n(k)} = \frac{(n-2k)(n-2k-1)}{2(k+1)} \tag{43}$$

可以看到,从 $k=0$ 直到当 k 近似于$\frac{1}{2}(n-\sqrt{n})$时,$t_n(k+1) \approx t_n(k)$,这些项逐渐地增加;然后当 k 超过$\frac{1}{2}n$ 时它们又逐渐地减少成为 0。主要的贡献显然来自 $k=\frac{1}{2}$

$(n-\sqrt{n})$的附近。通常都喜欢把主要贡献安排在值为 0 处,所以写

$$k=\frac{1}{2}(n-\sqrt{n})+x \tag{44}$$

并把 $t_n(k)$的大小看做 x 的函数。

摆脱 $t_n(k)$中阶乘的一个有用的方法是,使用斯特林近似公式,也就是等式 1.2.11.2-(18)。为此目的,把 x 限制在

$$-n^{\epsilon+1/4}\leqslant x\leqslant n^{\epsilon+1/4} \tag{45}$$

的范围内是方便的(我们马上就会见到),比如说,其中 $\epsilon=0.001$,使得可以把一个误差项包括进来。经过一个稍微麻烦的计算,作者曾经在 60 年代用手算它,但现在借助于计算机代数很容易就可完成,产生下列的公式

$$t_n(k)=\exp\Big(\frac{1}{2}\ln n-\frac{1}{2}n+\sqrt{n}-\frac{1}{4}\ln n-2x^2/\sqrt{n}-\frac{1}{4}-\frac{1}{2}\ln\pi-\frac{4}{3}x^3/n+2x/\sqrt{n}+\frac{1}{3}/\sqrt{n}-\frac{4}{3}x^4/n\sqrt{n}+O(n^{5\epsilon-3/4})\Big) \tag{46}$$

(45)中对 x 所做的限制是合理的,因为可以置 $x=\pm n^{\epsilon+1/4}$来给出所有被舍弃的项的上限,即

$$e^{-2n^{2\epsilon}}\exp\Big(\frac{1}{2}n\ln n-\frac{1}{2}n+\sqrt{n}-\frac{1}{4}\ln n-\frac{1}{4}-\frac{1}{2}\ln\pi+O(n^{3\epsilon-1/4})\Big) \tag{47}$$

而且如果乘以 n,则就得到所有被舍弃的项之和的上限。当 x 属于被限制的范围(45)时,这个上限的阶小于所要计算的各项的阶,因为因子 $\exp(-2n^{2\epsilon})$比起 n 的任何多项式来都要小得多。

显然,可以从这个和式中删去因式

$$\exp\Big(\frac{1}{2}n\ln n-\frac{1}{2}n+\sqrt{n}-\frac{1}{4}\ln n-\frac{1}{4}-\frac{1}{2}\ln\pi+\frac{1}{3}/\sqrt{n}\Big) \tag{48}$$

剩下的任务是在 $x=\alpha,\alpha+1,\cdots,\beta-2,\beta-1$ 的范围上对

$$\exp\Big(-2x^2/\sqrt{n}-\frac{4}{3}x^3/n+2x/\sqrt{n}-\frac{4}{3}x^4/n\sqrt{n}+O(n^{5\epsilon-3/4})\Big)=$$

$$\exp\left(\frac{-2x^2}{\sqrt{n}}\right)\left(1-\frac{4}{3}\frac{x^3}{n}+\frac{x^6}{n^2}\right)\left(1+2\frac{x}{\sqrt{n}}+2\frac{x^2}{n}\right)\times$$

$$\left(1-\frac{4}{3}\frac{x^4}{n\sqrt{n}}\right)(1+O(n^{9\epsilon-3/4})) \tag{49}$$

求和，其中 $-\alpha$ 和 β 近似地等于 $n^{\epsilon+1/4}$（不必是整数）。通过对求和区间的变换，欧拉求和公式，即等式 1.2.11.2-(10)，可以写成

$$\sum_{\alpha\leqslant x<\beta} f(x)=\int_\alpha^\beta f(x)\mathrm{d}x-\frac{1}{2}f(x)\bigg|_\alpha^\beta+\frac{1}{2}B_2\frac{f'(x)}{1!}\bigg|_\alpha^\beta+\cdots+$$

$$\frac{1}{m+1}B_{m+1}\frac{f^{(m)}(x)}{m!}\bigg|_\alpha^\beta+R_{m+1} \tag{50}$$

这里 $|R_m|\leqslant(4/(2\pi)^m)\int_\alpha^\beta|f^{(m)}(x)|\mathrm{d}x$。如果设 $f(x)=x^t\exp(-2x^2/\sqrt{n})$，其中 t 是一个固定的非负整数，欧拉求和公式将给出当 $n\to\infty$ 时 $\sum f(x)$ 的一个渐近级数，因为在这种情况下

$$f^{(m)}(x)=n^{(t-m)/4}g^{(m)}(n^{-1/4}x),\quad g(y)=y^t\mathrm{e}^{-2y^2} \tag{51}$$

且 $g(y)$ 是与 n 无关的一个很整齐的函数；导数 $g^{(m)}(y)$ 是 y 的多项式乘以 e^{-2y^2}，因此

$$R_m=O(n^{(t+1-m)/4})\int_{-\infty}^{+\infty}|g^{(m)}(y)|\mathrm{d}y=O(n^{(t+1-m)/4})$$

而且，如果在(50)的右边以 $-\infty$ 和 $+\infty$ 来代替 α 和 β，则在每项中至多造成阶为 $O(\exp(-2n^\epsilon))$ 的一个误差。于是

$$\sum_{\alpha\leqslant x<\beta} f(x)=\int_{-\infty}^{\infty}f(x)\mathrm{d}x+O(n^{-m}),\quad \text{对于所有 } m\geqslant 0 \tag{52}$$

给定 $f(x)$ 的这个特定的选择，仅仅这个积分是真正有意义的！这个积分不难求值（见习题 26），所以可以乘出公式(49)并求和，这给出

$$\sqrt{\pi/2}\left(n^{1/4}-\frac{1}{24}n^{-1/4}+O(n^{-1/2})\right)$$

于是，
$$t_n=\frac{1}{\sqrt{2}}n^{n/2}\mathrm{e}^{-n/2+\sqrt{n}-1/4}\left(1+\frac{7}{24}n^{-1/2}+O(n^{-3/4})\right) \tag{53}$$

实际上，在这里 O 项应该在指数中有一个额外的 9ϵ，但我们的处理方法使这样一点变得显然了，即如果在中间计算中取更高的精确性，则这个 9ϵ 就将消失。原则上，我们已经使用的这个方法可加以推广，以得到对于任意 k 的 $O(n^{-k})$，而不是

$O(n^{-3/4})$。t_n 的这一渐近级数，首先是由 Moser 和 Wyman 确定的（使用不同的方法），见*Canadian J. Math.* **7**(1955)，159～168。

我们用来推导(53)的方法对于渐近分析来说是极为有用的一项技术，它是由 P.S. Laplace[*Mémoires Acad. Sci.* (Paris，1782)，1～88]提出来的；详细的讨论请参见*CMath*，§9.4 中的"trading tails"。关于 tail-trading（尾部推销）的进一步的例子和推广，请参见 5.2.2 小节的结论。

习　题

1.[*16*] 什么图表(P,Q)对应于定理 A 的构造中的两行阵列

$$\begin{pmatrix}1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9\\ 6 & 4 & 9 & 5 & 7 & 1 & 2 & 8 & 3\end{pmatrix}$$

什么两行阵列对应于图表

$$P=\begin{array}{|c|c|c|}\hline 1 & 4 & 7\\ \hline 2 & 8 \\ \cline{1-2} 5 & 9 \\ \cline{1-2}\end{array}\qquad Q=\begin{array}{|c|c|c|}\hline 1 & 3 & 7\\ \hline 4 & 5 \\ \cline{1-2} 8 & 9 \\ \cline{1-2}\end{array}$$

2.[*M21*] 证明：当且仅当 t 是使得

$$p_{i_1}<p_{i_2}<\cdots<p_{i_t}=p,\quad q_{i_1}<q_{i_2}<\cdots<q_{i_t}=q$$

的下标 $i_1,i_2,\cdots,i_t$ 的最大个数时，(q,p)属于相对于(16)而言的类 t。

▶3.[*M24*] 证明在定理 A 的证明中，所定义的对应关系也可通过构造如下的一张表来得到

行 0	1	3	5	6	8
行 1	7	2	9	5	3
行 2	∞	7	∞	9	5
行 3		∞		∞	7
行 4					∞

这里行 0 和行 1 即是给定的两行阵列。对于 $k\geqslant 1$，行 $k+1$ 通过下列步骤由行 k 形成：

a)置 $p\leftarrow\infty$。

b)设第 j 列是具有下列性质的诸列中最左边的列，即：该列的第 k 行包含一个$<p$ 的整数，但第 $k+1$ 行是空白。如果不存在这样的列，且如果 $p=\infty$，则第 $k+1$ 行就完成了；如果不存在这样的列且 $p<\infty$，则返回 a)。

c)把 p 插入到第 $k+1$ 行的第 j 列中，然后置 p 等于第 k 行第 j 列处的项，并且返回 b。

一旦这张表以这样的方式构造出来，P 的第 k 行就由这张表的第 k 行中，那些不在该表的第$(k+1)$行中的整数所组成；Q 的第 k 行由在这张表的第 0 行中那样一些整数所组成，它们所在列的第 $k+1$ 行均含有∞。

▶4.[*M30*] 设 $a_1\cdots a_{j-1}a_j\cdots a_n$ 是不同元素的一个排列，并假定 $1<j\leqslant n$，如果

i)$j\geqslant 3$ 且 a_{j-2}处于 a_{j-1}和 a_j 之间，或者

ii)$j<n$ 且 a_{j+1}处于 a_{j-1}和 a_j 之间，

则由交换 a_{j-1}和 a_j 得到的排列 $a_1 \cdots a_{j-2}\, a_j\, a_{j-1}\, a_{j+1} \cdots a_n$ 称为“可允许的”。例如，对排列 1 5 4 6 8 3 7实施的可允许交换恰有 3 个；由于 1<4<5，我们可以交换 1 和 5；由于 3<6<8(或者由于 3<7<8)我们可以交换 8 和 3；但我们不能交换 5 和 4 或 3 和 7。

a)证明通过把元素 $a_1, a_2, \cdots a_n$ 逐次插入到开始时为空的图表，一个可允许的交换不改变由排列所形成的图表 P。

b)反之，证明通过一个或多个可允许的交换组成的一个序列，有相同的 P 图表的任何两个排列可彼此进行转换[提示：给定 P 的形状是$(n_1, n_2, \cdots, n_m)$。试证明对应于 P 的任何排列，通过一个可允许交换的序列，可被转换成“规范排列”$p_{m1} \cdots p_{mn_m} \cdots p_{21} \cdots p_{2n_2} p_{11} \cdots p_{1n_1}$]。

▶5.[*M22*] 设 P 是对应于排列$a_1\, a_2 \cdots a_n$ 的图表；利用习题 4 证明 P^{T} 是对应于 $a_n \cdots a_2\, a_1$ 的图表。

6.[*M26*](M.P.Schützenberger) 设 π 是具有k 个不动点的一个对合，证明在定理 B 的推论的证明中，对应于 π 的图表恰有 k 个奇数长度的列。

7.[*M20*](C.Schensted) 设 P 是对应于排列$a_1 a_2 \cdots a_n$ 的图表。证明 P 中列的数目是一个递增子序列$a_{i_1} < a_{i_2} < \cdots < a_{i_c}$的最大长度$c$，其中 $i_1 < i_2 < \cdots < i_c$；P 中行的数目是一个递减子序列 $a_{j_1} > a_{j_2} > \cdots > a_{j_r}$的最大长度 r，其中 $j_1 < j_2 < \cdots < j_r$。

8.[*M18*][P.Erdös, G.Szekeres] 证明含有多于 n^2 个元素的任何排列，都有一个长度大于 n 的单调子序列，但却有 n^2 个元素的排列，且它无长度大于 n 的单调子序列[提示：见前一习题]。

9.[*M24*] 继续习题 8，对于没有长度大于 n 的单调子序列的$\{1,2,\cdots,n^2\}$的排列，求出一个“简单的”精确个数的公式。

10.[*M20*] 证明若 P 开始时是一图表，则当算法 S 终止时，P 仍是一图表。

11.[*20*] 仅仅给定算法 S 终止后的 r 和 s 之值，是否有可能按原来的条件恢复 P？

12.[*M24*] 如果反复地使用算法 S，删去其形状如$(n_1, n_2, \cdots, n_m)$的一个图表 P 的所有元素，问步骤 S3 被执行多少次？对于 $n_1 + n_2 + \cdots + n_m = n$ 的所有形状，这个量的最小值是多少？

13.[*M28*] 证明定理 C。

14.[*M43*] 找出定理 D c)部分的一个更直接的证明。

15.[*M20*] 多重集合$\{l \cdot a, m \cdot b, n \cdot c\}$有多少具有这样性质的排列，即当从左到右读这个排列时，c 的数目决不超过b 的数目，而 b 的数目决不超过a 的数目(例如，$a\,a\,b\,c\,a\,b\,b\,c\,a\,c\,a$ 是这样一个排列)？

16.[*M08*] 由(39)表示的偏序可以用多少种方式按拓扑排序？

17.[*HM25*] 设

$$g(x_1, x_2, \cdots, x_n; y) = x_1\Delta(x_1 + y, x_2, \cdots, x_n) + x_2\Delta(x_1, x_2 + y, \cdots, x_n) + \cdots + x_n\Delta(x_1, x_2, \cdots, x_n + y)$$

证明

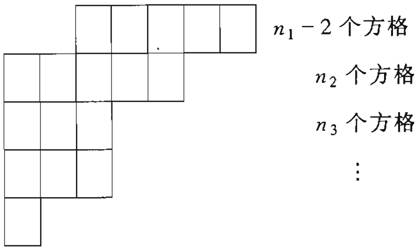
$$g(x_1, x_2, \cdots, x_n; y) = \left(x_1 + x_2 + \cdots + x_n + \binom{n}{2} y\right)\Delta(x_1, x_2, \cdots, x_n)$$

[提示：多项式 g 是齐次的(所有的项都有相同的总次数)。而且，它是对诸 x 反对称的(交换 x_i 和x_j 会改变g 的符号)]。

18.[*HM30*] 推广习题 17，当 $m \geqslant 0$ 时计算和式

$$x_1^m\Delta(x_1 + y, x_2, \cdots, x_n) + x_2^m\Delta(x_1, x_2 + y, \cdots, x_n) + \cdots + x_n^m\Delta(x_1, x_2, \cdots, x_n + y)$$

19.[*M40*] 求填写一个阵列的方式种数的公式,这个阵列和一个图表一样,但在行 1 的左边除去两个方格;例如



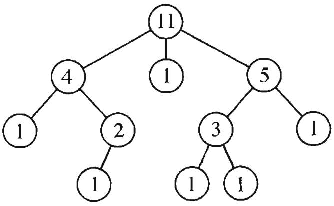
就是这样一个形状(行和列,如同在通常的图表中那样,是按递增顺序排列的)。

换言之,$\{1,2,\cdots,n_1+\cdots+n_m\}$上形状为

$$(n_1,n_2,\cdots,n_m)$$

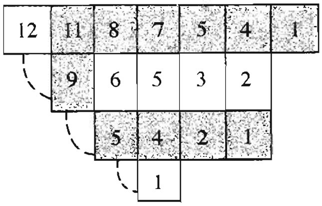
的图表中,头行兼有元素 1 和元素 2 者有多少?

▶20.[*M24*] 对具有元素$\{1,2,\cdots,n\}$的一个给定二叉树的节点进行标号,使得每个节点的标号小于它的后裔的标号。证明这样加标号的方式种数,是 $n!$除以“子树的长度”(即在每个子树中节点的个数)的乘积。例如,对



的节点进行标号的方式种数是 $11!/11\cdot4\cdot1\cdot5\cdot1\cdot2\cdot3\cdot1\cdot1\cdot1\cdot1=10\cdot9\cdot8\cdot7\cdot6$(参考定理 H)。

21.[*HM31*](R.M.Thrall) 设 $n_1>n_2>\cdots>n_m$ 确定一个“移位图表”的形状,其中行 $i+1$ 从行 i 的右边一个位置开始。例如,形状(7,5,4,1)的移位图表有图式



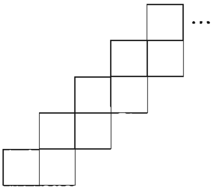
试证明,把整数 $1,2,\cdots,n=n_1+n_2+\cdots+n_m$ 安排到形如$(n_1,n_2,\cdots,n_m)$的移位图表,使得行和列都有递增次序的方式数,是 $n!$除以“广义钩子长度”的乘积;在上边的图式中,已经用阴影标出对应于 1 行 2 列方格的长度为 11 的广义钩子(阵列左边“倒楼梯”部分中的钩子,其形状为旋转 90°的 U 形,而不是一个 L 形)。因此共有

$$17!/12\cdot11\cdot8\cdot7\cdot5\cdot4\cdot1\cdot9\cdot6\cdot5\cdot3\cdot2\cdot5\cdot4\cdot2\cdot1\cdot1$$

种方式,以行和列处于递增次序来填满这个形状。

22.[*M39*] 以集合$\{1,2,\cdots,N\}$的元素填入形为$(n_1,n_2,\cdots,n_m)$的一个阵列,并且允许重复。问能有多少种方式,使得行是非减的,而列是严格递增的?例如简单的 m 行图形$(1,1,\cdots,1)$可以$\binom{N}{m}$种方式来填入;1 行的形状(m)可以$\binom{m+N-1}{m}$种方式填入;图形$(2,2)$可以$\frac{1}{3}\binom{N+1}{2}\binom{N}{2}$种方式填入。

▶23.[*HM30*](D. Andre) 把数$\{1,2,\cdots,n\}$放置到 n 个格子的阵列



中去,使得行和列处于递增的次序,问有多少种方式 A_n? 求生成函数 $g(z)=\sum A_n z^n/n!$。

24.[*M28*] 证明

$$\sum_{\substack{q_1+\cdots+q_n=t\\0\leqslant q_1,\cdots,q_n\leqslant m}}\binom{m}{q_1}\cdots\binom{m}{q_n}\Delta(q_1,\cdots,q_n)^2=$$

$$n!\binom{nm-(n^2-n)}{t-\frac{1}{2}(n^2-n)}\binom{m}{n-1}\binom{m}{n-2}\cdots\binom{m}{0}\Delta(n-1,\cdots,0)^2$$

[提示:证明 $\Delta(k_1+n-1,\cdots,k_n)=\Delta(m-k_n+n-1,\cdots,m-k_1)$;以类似于(38)的方式分解一个 $n\times(m-n+1)$的图表;并同推导(36)中那样处理这个和式。]

25.[*M20*] 为什么(42)是对合的生成函数?

26.[*HM21*] 当 t 是一个非负整数时,计算$\int_{-\infty}^{\infty}x^t\exp(-2x^2/\sqrt{n})\mathrm{d}x$。

27.[*M24*] 设 Q 是$\{1,2,\cdots,n\}$上的杨氏图表,元素 i 在行r_i 和列c_i 中。当 $r_i<r_j$ 时,我们说 i 在j"之上"。

a)证明:对于 $1\leqslant i<n$,当且仅当 $c_i\geqslant c_{i+1}$时 i 在$i+1$ 之上。

b)给定 Q 使得(P,Q)对应于排列

$$\begin{pmatrix}1&2&\cdots&n\\a_1&a_2&\cdots&a_n\end{pmatrix}$$

证明 i 在$i+1$ 之上,当且仅当 $a_i>a_{i+1}$(因此只要知道 Q,就能确定排列中路段的个数。这一结果属于 M.P. Schützenberger)。

c)证明,对于 $1\leqslant i<n$,在 Q 中i 在$i+1$ 之上,当且仅当在 Q^S 中$i+1$ 是在 i 之上。

28.[*M43*] 证明$\{1,2,\cdots,n\}$的一个随机排列的最长递增子序列的平均长度近似于 $2\sqrt{n}$(其对应于定理 A 中行 1 的平均长度)。

29.[*HM25*] 试证明 n 个元素的一个随机排列,有长度$\geqslant l$ 的一个递增子序列的概率$\leqslant\binom{n}{l}/l!$。当 $l=\mathrm{e}\sqrt{n}+O(1)$时,这个概率是 $O(1/\sqrt{n})$;而当 $l=3\sqrt{n}$,$c=6\ln 3-6$ 时,这个概率是 $O(\exp(-c\sqrt{n}))$。

30.［*M41*］(M.P.Schützenberger)证明从 P 到 P^S 所进行的操作,是应用于任何有限偏序集合(而不仅仅是图表)的一个操作的特殊情况:用整数$\{1,2,\cdots,n\}$来对一个偏序集的元素进行标号,使得偏序同标号相一致。用类似于算法 S 的方式移动其它标号,同时逐次地删去标号 1,2,…,并且把 1,2,…放置在空了的位置上,试求类似于(26)的一个对偶标号。证明当以相反的数值次序重复对偶标号时,这个操作产生原来的标号。另外请剖析这个操作的其它性质。

31.［*HM30*］设 x_n 为在一个 $n\times n$ 的棋盘上,放置 n 个互不冲突的车的方式数,其中每种放置通过两个对角线的反射均保持不变。于是有 $x_4=6$(对合被要求仅关于一个对角线是对称的。习题 5.1.3-19 考虑了一个相关的问题)。试求 x_n 的渐近特性。

32.［*HM21*］试证明当 X 为有均值 1 和方差 1 的标准离差时,t_n 是 X^n 的期望值。

33.［*M25*］(O.H.Mitchel,1881) 真或假:当 $a_1,a_2,\cdots,a_n$ 是整数时,$\Delta(a_1,a_2,\cdots,a_n)/\Delta(1,2,\cdots,m)$是一个整数。

34.［*25*］(T.Nakayama,1940) 试证明,如果一个图表形包含长度为 ab 的一个钩子,则它也包含长度为 a 的一个钩子。

35.［*30*］(A.P.Hillman 和 R.M.Grassl,1976) 当行 i 有 n_i 个格子,列 j 有 n_j' 个格子时,如果 $\sum p_{ij}=m$ 且

$$p_{i1}\geqslant\cdots\geqslant p_{in_i},p_{1j}\geqslant\cdots\geqslant p_{n_j'j},\text{对于 }1\leqslant i\leqslant n_1',1\leqslant j\leqslant n_1$$

则在一个图表形中对非负整数 p_{ij} 的一个安排,称做 m 的一个**平面分划**。而如果代之以

$$p_{i1}\leqslant\cdots\leqslant p_{in_i},p_{1j}\leqslant\cdots\leqslant p_{n_j'j},\text{对于 }1\leqslant i\leqslant n_1',1\leqslant j\leqslant n_1$$

则称它为**逆平面分划**。

考虑下列算法,它实现对于一个给定形状的逆平面分划,并且构造有相同形状的数 q_{ij} 的另一个数组:

G1.［初始化］对于 $1\leqslant j\leqslant n_i$ 和 $1\leqslant i\leqslant n_1'$,置 $q_{ij}\leftarrow 0$。然后置 $j\leftarrow 1$。

G2.［求非零的格子］如果 $p_{n_j'j}>0$,置 $i\leftarrow n_j'$,$k\leftarrow j$,并继续步骤 G3。否则,如果 $j<n_1$,则 j 增加 1 并重复此步骤。否则停止(p 数组现在为零)。

G3.［P 减少］p_{ik}减 1。

G4.［向上或向右移动］如果 $i>1$,且 $p_{(i-1)k}>p_{ik}$,则 i 减 1,并返回 G3。否则如果 $k<n_i$,则 k 加 1 并返回 G3。

G5.［增加 q］q_{ij}加 1 并返回 G2。 ▌

通过设计从诸 q 重新计算诸 p 的算法,试证明,这一构造定义 m 的逆平面分划和方程

$$m=\sum h_{ij}q_{ij}$$

的解之间的一一对应,其中数 h_{ij}是这个形状的钩长。

36.［*HM27*］(R.P.Stanley,1971) (a)试证明,在一个给定的形状中,m 的逆平面分划的个数为$[z^m]1/\prod(1-z^{h_{ij}})$,其中数 h_{ij}是这个形状的钩长。(b)由这个结果推导定理 H[提示:当 $m\to\infty$时什么是诸分划的渐近个数]。

37.［*M20*］(P.A.MacMahon,1912) 什么是所有平面分划的生成函数(当图表的形状为无限时,z^m 的系数应为 m 的平面分划的总数)?

38.［*M30*］(Green,Nijenhuis 及 Wilf,1979) 通过令有向边从每个格子通到它钩子中的其它格子,在任何给定的图表形状的格子 T 上,我们可以构造一个有向无循环的图;这样格子(i,j)的出度(Outdegree)将为 $d_{ij}=h_{ij}-1$,其中 h_{ij}为钩子的长度。假设通过选择一个随机的起始格子(i,j)

并且随机选择进一步的有向边，我们在这个有向图中生成一个随机通路，直到不再有出口的一个角落格子为止。每个随机选择被一致地作成：

a)令(a,b)是 T 的一个角落格子，并设 $I=\{i_0,\cdots,i_k\}$和 $J=\{j_0,\cdots j_l\}$是满足 $i_0<\cdots<i_k=a$ 和$j_0<\cdots<j_l=b$ 的行和列的集合。这个有向图包含$\binom{k+l}{k}$条通路，其行和列的集合分别为 I 和 J；令 $P(I,J)$为随机通路是这些通路之一的概率。试证明

$$P(I,J)=1/(nd_{i_0b}\cdots d_{i_{k-1}b}d_{aj_0}\cdots d_{aj_{l-1}})$$

其中 $n=|T|$。

b)设 $f(T)=n!/\Pi h_{ij}$。试证明随机通路在角落(a,b)处结束的概率为 $f(T\setminus\{(a,b)\})/f(T)$。

c)证明，b)的结果证明了定理 H，也给了我们一个方法，来生成形状 T 的一个随机图表，而且所有 $f(T)$个图表都同样可能。

39.[*M38*] (I.M.Pak 和 A. V. Stoyanovskii，1992) 设 P 是整数$\{1,\cdots,n\}$的任何排列所填满的形状为$(n_1,\cdots,n_m)$的数组，其中 $n=n_1+\cdots+n_m$。下列的过程类似于 5.2.3 小节的"上移"算法，可以用来把 P 转换为一个图表。它也定义了一个同样形状的数组 Q，可以用来提供定理 H 的一个组合证明。

P1.[对(i,j)进行循环] 以词典排序的逆序，对数组的所有格子(i,j)实施步骤 P2 和 P3(即从底向上，且在每行从右向左)；然后停止。

P2[在(i,j)处固定 P] 置 $K\leftarrow P_{ij}$，并且实施算法 S′(见下文)。

P3[调整 Q] 对于 $j\leqslant k<s$，置 $Q_{ik}\leftarrow Q_{i(k+1)}+1$，并置 $Q_{is}\leftarrow i-r$。 ▎

这里算法 S′和 Schützenberger 的算法 S 相同，只是步骤 S1 和 S2 被稍微推广：

S1′[初始化] 置 $r\leftarrow i,s\leftarrow j$。

S2′[完成没有?]　如果 $K\lesssim P_{(r+1)s}$和$K\lesssim P_{r(s+1)}$，置 $P_{rs}\leftarrow K$ 并终止。

(算法 S 实质上是 $i=1,j=1,K=\infty$的特殊情况。)

例如，算法 P 把形如(3,3,2)的一个特殊数组以下列方式弄直，如果我们在步骤 P2 的开始处来观察数组 P 和 Q 的内容，而且以 P_{ij}为粗体的字体：

	7	8	5		7	8	5		7	8	5		7	8	5		7	8	5		7	8	**5**		7	**8**	4		**7**	3	4
$P=$	1	6	4		1	6	4		1	6	**4**		1	**6**	4		**1**	3	4		1	3	4		1	3	5		1	5	8
	3	**2**			**3**	2			2	3			2	3			2	6			2	6			2	6			2	6	
																											−1			0	−1
$Q=$															0			−1	0		0	−1	0		0	−1	0		0	−1	0
						0			1	0			1	0			1	0			1	0			1	0			1	0	

最后的结果是

	1	3	4			1	−2	−1
$P=$	2	5	8		$Q=$	0	−1	0
	6	7				1	0	

a)如果 P 只不过是一个 $1\times n$ 的数组，算法 P 把它排序成为$\boxed{1\ \cdots\ n}$。试说明在这种情况下 Q 数组将包含什么?

b)如果 P 是$n\times1$ 而不是 $1\times n$，试回答相同的问题。

c)证明，在一般情况下，我们将有

$$-b_{ij} \leqslant Q_{ij} \leqslant r_{ij}$$

其中 b_{ij}是(i,j)之下的格子数，而 r_{ij}是右边的格子数。于是，对于 Q_{ij}可能的值的个数恰是h_{ij}，即第(i,j)个钩子的大小。

d)如果我们能够证明算法 P，定义了以 $n!$种方法填入原来的形状，和输出数组(P,Q)的对偶之间的一一对应(其中 P 是一个图表，Q 的元素满足 c)的条件)，那么就可以建设性地证明定理 H。因此我们要求算法 P 的一个逆。对于什么样的初始排列，算法 P 产生 2×2 的数组 $Q=\begin{pmatrix}0 & -1\\ 0 & 0\end{pmatrix}$。

e)什么样的初始排列，根据算法 P 可转换成下列数组？

$P=$

1	3	5	7	11	15
2	6	8	14		
4	9	13			
10	12				
16					

，

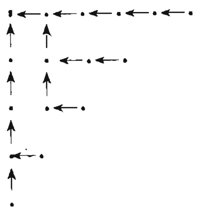
$Q=$

−2	−3	−1	−1	1	0
3	−2	−1	0		
0	−1	0			
−1	0				
0					

f)根据给定数组(P,Q)的任何对偶(其中 P 是一个图表，Q 满足 c)的条件)，试设计算法 P 的反序算法[提示：构造一棵有向树，其顶点是格子(i,j)，并有有向边

$$(i,j)\to(i,j-1)\qquad 如果\ P_{i(j-1)} > P_{(i-1)j}$$
$$(i,j)\to(i-1,j)\qquad 如果\ P_{i(j-1)} < P_{(i-1)j}$$

在 e)的例子中，我们有树



这个树的诸通路显示了反序算法 P 的键码]。

40.[*HM43*] 假设通过逐次地以这样一个方式来放置数 $1,2,\cdots,n$，即当放置一个新数时，每个可能性都是相同可能的。例如，利用这一过程，图表(1)将以$\frac{1}{1}\cdot\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{3}\cdot\frac{1}{3}\cdot\frac{1}{3}\cdot\frac{1}{4}\cdot\frac{1}{3}\cdot\frac{1}{3}\cdot\frac{1}{4}\cdot\frac{1}{4}\cdot\frac{1}{5}\cdot\frac{1}{4}\cdot\frac{1}{5}\cdot\frac{1}{4}$的概率被得到。

试证明，对于 $O\leqslant k\leqslant m$，得到的形状$(n_1,n_2,\cdots,n_m)$将以很高的概率有 $m\approx\sqrt{6n}$ 和$\sqrt{k}+\sqrt{n_{k+1}}\approx\sqrt{m}$。

41.[*25*] (图书馆中的无序问题)粗心的读者经常把书架上的书放回到错误的位置上。计算一个图书馆中存在的无序数量的一个方式是，考虑在把所有的图书恢复到正确的位置之前，需要把一本书从一个位置取出并且插入到另一个位置的极小次数。

于是设 $\pi=a_1a_2\cdots a_n$ 是$\{1,2,\cdots,n\}$的一个排列，一个“删去-插入”操作对于某个 i 和 j 把 π 变成

$$a_1\cdots a_{i-1}a_{i+1}\cdots a_ja_ia_{j+1}\cdots a_n\qquad 或\qquad a_1\cdots a_ja_ia_{j+1}\cdots a_{i-1}a_{i+1}\cdots a_n$$

令 dis(π)是将把 π 排成有序的“删去-插入”操作的极小次数。能否借助于 π 的较简单的特征来表达 dis(π)?

▶42. [*30*](基因组中的无序问题) Lobelia fervens DNA 有以 $g_7^R g_1 g_2 g_4 g_5 g_3 g_6^R$ 的序列出现的基因,其中 g_7^R 代表 g_7 从左至右的反射;同样的基因出现在烟草植物中,但次序为 $g_1g_2g_3g_4g_5g_6g_7$。证明为了从 $g_1g_2g_3g_4g_5g_6g_7$ 得到 $g_7^R g_1 g_2 g_4 g_5 g_3 g_6^R$,对于子串需要 5 次“触发”操作(当 α,β 和 γ 为串时,一个触发把 $\alpha\beta\gamma$ 变成 $\alpha\beta^R\gamma$)。

43. [*35*] 继续上一题,试证明,为了对 $g_1g_2\cdots g_n$ 的任何重新排列(Rearrangement)进行排序,最多需要 $n+1$ 次触发。对于所有 $n>3$,试构造需要 $n+1$ 次触发的例子。

44. [*M37*] 如果所有 $2^n n!$ 个基因组的重排都是同样可能的,试证明,为对 n 个基因的一个随机排列进行排序,所要求触发的平均数大于 $n-H_n$。

5.2　内部排序

让我们通过一项小实验,来开始有关好的“排序者队伍”的讨论。你将如何解决下面的程序设计问题?

“存储单元 $R+1$, $R+2$, $R+3$, $R+4$ 和 $R+5$ 包含 5 个数,试写出一个计算机程序,使得当必要时,它重新把这些数排成为递增次序的。”(如果你已经熟悉某些排序方法,就请你尽量暂时忘掉它们;想像你头一次着手解决这个问题,没有任何关于怎样做的预先的知识。)

在进一步往下阅读之前,要求你构造一个这个问题的解。

……………………………………………………………………………………

在解决上述这个具有挑战性问题时所花费的时间,将在你阅读这一章时获得补偿。可能,你的解是下列类型之一:

A. *一个插入排序*　逐个考察诸项,每一个新的项被插入到相对于以前已排好序的诸项的适当位置上(这是许多玩桥牌者每抓一张牌时,对他们手中的牌进行排序的一种方式)。

B. *一个交换排序*　如果发现某两项次序颠倒,则交换它们。重复这一过程直到不需要再作进一步的交换为止。

C. *一个选择排序*　首先找到最小的(或最大的)项,并设法把它同余下的分开来;然后选择下一个最小的(或最大的),等等。

D. *一个枚举排序*　每个项都同其它项进行比较;一个项的最后位置由它超过的键码(Key)的个数确定。

E. *一个专用排序*　如同在本问题中所述的那样,它对 5 个元素进行排序,可能工作得很好,但却不能容易地推广到较大的项数。

F. *一种懒散的态度*　在这种态度下你忽略了上面的建议,并且决定全然不去解决这个问题。对不起。现在你已经读得太远了,而且已经错过了时机。

G. *一项新的超级排序技术*　它是对已知方法的明显改进(如果有,请立刻通知作者)。

如果这个问题比如说有 1000 项，而不只是 5 项，你可能已经发现了将在后边述及的更为精巧的技术。但无论如何，当着手解决一个新问题时，先找某些相当明显的解决方法，然后再试图来改进它，往往是明智的。上述的 A、B 和 C 引出了几类重要的排序技术，它们是这些简单想法的深化。

目前已经发明了许多不同的排序算法，我们将在本书讨论其中大约 25 个算法。这样颇使人惊讶的众多的方法，实际上还只是迄今已经想出的算法的一小部分；在我们的讨论中，将略去许多现在已被废弃的方法，或者仅仅简单地提及它们。为什么会有这么多的排序方法呢？在计算机的程序设计中，常有“为什么会有这样多的 x 方法呢?”的问题，其中 x 就是某个问题的集合。这个问题的答案是：每种方法都有它的优点和缺点，对于某些数据和硬件配置来说，它就有可能超过其它的方法。可惜，还不知道“最好的”排序方法；目前许多最好的方法，都是针对特定的机器，根据特定的目的，对特定对象进行排序所得到的。用 Rudyard Kipling 的话说：“有 69 种进行部落安置的方式，而且它们每一种都是对的。”

一个好的想法是学习每种排序方法，它能帮助你具体的应用做出明智的选择。幸而，学习这些算法并不是一项艰难的任务，因为它们都以有趣的方式相互关联着。

在本章开始时，已定义了将在排序研究中使用的基本术语和符号：记录

$$R_1, R_2, \cdots, R_N \tag{1}$$

有待按其键码 $K_1, K_2, \cdots, K_N$ 的非减次序进行排序，实质上要求找出一个排列 $p(1)p(2)\cdots p(N)$，使得

$$K_{p(1)} \leqslant K_{p(2)} \leqslant \cdots \leqslant K_{p(N)} \tag{2}$$

在本节中，我们讨论*内部排序*。此时，有待排序的记录个数足够小，以致整个过程都能在一台计算机的高速存储器中实现。

在某些情况下，会要求在存储器中对这些记录物理地重新排列，使得它们的键码按次序排列。但在另外的情况下，则可能只要指明这个排列的某个辅助表就够了。如果每个记录和/或键码要占用相当多的计算机存储器，则构造一个新的指向记录的链接地址表，并处理这些链接地址，而不是到处移动庞大的记录，通常更好些。这种方法称为*地址表排序*(见图 6)。如果键码很短，但是记录的附属信息很长，则为了获得更高的速度，这个键码即可用作链接地址，这就是所谓*键码排序*。另一种排序方案利用了包括在每个记录中的一个辅助链接字段；链接的方式是使这些记录最终被链接在一起以形成一个直接的线性表，每个链接指向下一个记录，这就是所谓*表排序*(见图 7)。

在用地址表方法或表方法进行排序之后，诸记录可像所希望的那样，重新排成递增的顺序。习题 10 和 12 讨论了做这件事的有趣方法，只要求有足够容纳一个记录的附加存储空间即可；或者，可以简单地把这些记录都移到一个能容纳所有记录的新区域。后一方法通常比头一个方法快两倍，但它几乎要求两倍的存储空间。在许多应用中，全然不需要移动记录，因为对于随后的寻址操作而言，使用链接字段通常已足够了。

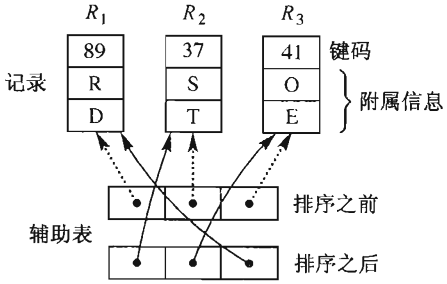


图 6　地址表排序

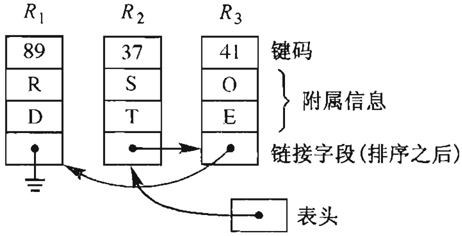


图 7　表排序

我们将通过 4 个方面来说明将要深入讨论的所有排序方法,即

a)算法的一个英语语言描述;

b)一个框图;

c)一个 MIX 程序;

d)一个排序方法的示例,它应用于某个 16 个数的集合。

为了方便起见,MIX 程序通常都假定键码是数值的,并且能放到一个单字中去;有时,甚至把键码限制为一个字的一部分。次序关系"<"将是通常的算术次序;记录将只由键码组成,而没有附属的信息。这些假定使得程序更短和更容易理解。读者应当发现,使用地址表排序或表排序,能很容易地把程序改成一般通用的情况。对每个排序算法运行时间的分析,将通过 MIX 程序进行。

通过计数进行排序　作为研究内部排序方法的一个简单示例,考虑在本节开头提出的"计数"思想。这个简单的方法是以这样一个思想为基础的,即在最后排好序的序列中,第 j 个键码恰恰大于 $(j-1)$ 个其它键码。换言之,如果知道某个键码确实超过 27 个其它键码,而且没有两个键码相同,则在排序之后对应的记录应当进入位置 28。所以,这个思想是比较每对键码,计算有多少个键码小于每一个特定的键码。

进行这些比较的明显方法是

$$\text{对于 } 1 \leqslant i \leqslant N(\text{对于 } 1 \leqslant j \leqslant N(\text{比较 } K_j \text{ 和 } K_i))$$

但容易看出,这些比较中有一半以上是多余的,因为没有必要把一个键码同它自己进行比较,也没有必要比较 K_a 和 K_b 然后比较 K_b 和 K_a。我们只需要比较

$$\text{对于 } 1 < i \leqslant N(\text{对于 } 1 \leqslant j < i(\text{比较 } K_j \text{ 和 } K_i))$$

因此导出了下列算法。

算法 C(比较计数)　本算法通过维护一张辅助表 COUNT[1],…,COUNT[N],对于小于一个给定键码的键码个数进行计数,来实现用键码 $K_1,\cdots,K_N$ 对记录 $R_1,\cdots,R_N$ 进行排序。算法结束时,COUNT[j]+1 确定记录 R_j 的最后位置。

C1.[清空 COUNT]　把 COUNT[1]至 COUNT[N]都置成 0。

C2.［对 i 进行循环］ 对 $i=N,N-1,\cdots,2$ 实施步骤 C3；然后结束此算法。

C3.［对 j 进行循环］ 对 $j=i-1,i-2,\cdots,1$ 实施步骤 C4。

C4.［比较 $K_i:K_j$］ 如果 $K_i<K_j$，则 COUNT[j]加 1，否则 COUNT[i]加 1。▌

注意，此算法不涉及记录的移动。它类似于地址表排序，因为 COUNT 表确定了这些记录最后的安排；但是由于 COUNT[j]告诉我们往何处移动 R_j，而不是指出哪一个记录应当被移动到 R_j 的位置，故它与地址表排序略有不同(因此在 COUNT 表中确定了排列 $p(1),\cdots,p(N)$的逆，见 5.1.1 小节)。

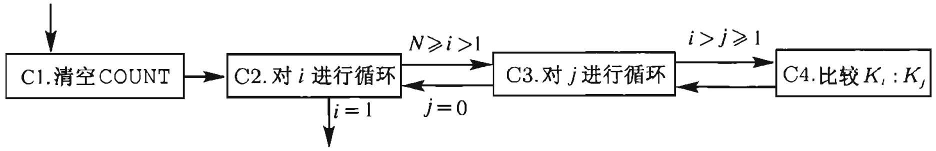


图 8 算法 C：比较计数

通过把比较计数算法应用于作者 1963 年 3 月 19 日随机地选择的 16 个数上，表 1 说明了这个算法的典型特性。同样的 16 个数将用于说明我们后面将要讨论的绝大多数方法。

表 1 计数排序(算法 C)

键码	503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703
COUNT(init.):	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
COUNT(i=N)	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	12
COUNT(i=N-1):	0	0	0	0	2	0	2	0	0	0	0	0	0	0	13	12
COUNT(i=N-2):	0	0	0	0	3	0	3	0	0	0	0	0	0	11	13	12
COUNT(i=N-3):	0	0	0	0	4	0	4	0	1	0	0	0	9	11	13	12
COUNT(i=N-4)	0	0	1	0	5	0	5	0	2	0	0	7	9	11	13	12
COUNT(i=N-5):	1	0	2	0	6	1	6	1	3	1	2	7	9	11	13	12
......																
COUNT(i=2):	6	1	8	0	15	3	14	4	10	5	2	7	9	11	13	12

在描述这个算法以前所进行的讨论中，我们轻率地假定没有任何两个键码相等。这个假定有潜在的危险，因为如果相等的键码对应于相等的 COUNT，则这些记录的最后重新安排将是十分复杂的。幸而，如同习题 2 所示，不管出现多少相等的键码，算法 C 仍能给出正确的结果。

程序 C(比较计数) 以下是算法 C 的 MIX 实现。假定对于 $1\leqslant j\leqslant N$，R_i 存于单元 INPUT + j 中，而 COUNT[j]存于单元 COUNT + j 中；$rI1\equiv i$；$rI2\equiv j$；$rA\equiv K_i\equiv R_i$；

rX≡COUNT[i]。

```
START  ENT1  N         1      C1.清空 COUNT
       STZ   COUNT,1   N      COUNT[i]←0
       DEC1  1         N
       J1P   *-2       N      N≥i>0
       ENT1  N         1      C2.对 i 进行循环
       JMP   1F        1
2H     LDA   INPUT,1   N-1
       LDX   COUNT,1   N-1
3H     CMPA  INPUT,2   A      C4.比较 Ki:Kj
       JGE   4F        A      如果 Ki≥Kj 则转移
       LD3   COUNT,2   B      COUNT[j]
       INC3  1         B        +1
       ST3   COUNT,2   B      →COUNT[j]
       JMP   5F        B
4H     INCX  1         A-B    COUNT[i]←COUNT[i]+1
5H     DEC2  1         A      C3.对 j 循环
       J2P   3B        A
       STX   COUNT,1   N-1
       DEC1  1         N-1
1H     ENT2  -1,1      N      N≥i>j>0
       J2P   2B        N      ▌
```

这个程序的运行时间是 $13N+6A+5B-4$ 个单位，其中 N 是记录个数；A 是从 N 个对象的集合中选择两个的数目，即 $\binom{N}{2}=(N^2-N)/2$；B 是满足 $j<i$ 且 $K_j>K_i$ 的下标对偶的数目。因此，B 是排列 $K_1,\cdots,K_N$ 的*反序数*；这是在 5.1.1 小节被深入分析过的量，在等式 5.1.1-(12)和 5.1.1-(13)中，我们发现对于随机次序下的不相等的键码，我们有

$$B=(\min 0,\text{ave}(N^2-N)/4,\max(N^2-N)/2,\text{dev}\sqrt{N(N-1)(N+2.5)}/6)$$

因此程序 C 要求的时间单位在 $3N^2+10N-4$ 和 $5.5N^2+7.5N-4$ 之间，而平均运行时间在这两个极值的正中间。例如，表 1 中的数据有 $N=16,A=120,B=41$，所

以程序 C 将用 1129 个单位时间进行排序。关于程序 C 的一个改型，请看习题 5，它有稍微不同的时间特征。

在这个运行时间中占支配地位的因子 N^2 说明，当 N 很大时，算法 C 不是一个有效的方法。记录数加 1 倍，就会使运行时间增加 3 倍。由于这个方法要求比较所有不同的键码对偶(K_i,K_j)，因此没有显而易见的方法来使它摆脱对于 N^2 的依赖性，尽管在这一章稍后将看到，利用"其它"技术，排序的最坏情况的运行时间可以减少到 $N\log N$。我们对于算法 C 的主要兴趣在于它的简便性，而不在于它的速度；算法 C 作为一个示例，表明了将要描述的更为复杂（和更为有效）的方法的风格。

通过计数进行排序，还有另外一个方法，从有效性的观点看，它是十分重要的；它主要应用于有许多相同的键码出现，且所有的键码都落入范围 $u\leqslant K_j\leqslant v$ 的情况，其中$(v-u)$很小。这些假定看来是十分严格的限制，但是事实上将看到这一思想有不少的应用。例如，如果把这个算法应用于键码的头几位数字，而不是整个键码，则这个文件将被部分地排序，而且完成这项任务将相当的简单。

为了了解其中蕴涵的原理，假设所有的键码位于 1 和 100 之间。当第一遍扫描这个文件时，可以统计有多少个 1，2，…，100 出现；而在第二遍扫描时，就可以把这些记录移到输出区域中的适当位置。下列算法给出此过程的完整细节。

算法 D（分布计数） 假定所有键码都是 $u\leqslant K_j\leqslant v$ 范围中的整数，其中 $1\leqslant j\leqslant N$。本算法利用一张辅助表 COUNT[u]，…，COUNT[v]对记录 $R_1,\cdots,R_N$ 进行排序。算法结束时，这些记录以所希望的次序移到一个输出区域 $S_1,\cdots,S_N$ 中去。

D1.［清空 COUNT］ 把 COUNT[u]至 COUNT[v]全部清成 0。

D2.［对 j 进行循环］ 对于 $1\leqslant j\leqslant N$ 实施步骤 D3；然后转到 D4。

D3.［COUNT[K_j]增值］ COUNT[K_j]的值增 1。

D4.［累加］ （这时 COUNT[i]是等于 i 的键码的个数。）对于 $i=u+1,u+2,\cdots,v$，置 COUNT[i]←COUNT[i]+COUNT[$i-1$]。

D5.［对 j 进行循环］ （这时 COUNT[i]是小于或等于 i 的键码的个数，特别是 COUNT[v]$=N$。）对 $j=N,N-1,\cdots,1$ 实施步骤 D6； 然后终止这个算法。

D6.［输出 R_j］ 置 i←COUNT[K_j]，$S_i\leftarrow R_j$ 及 COUNT[K_i]←$i-1$。 ▌

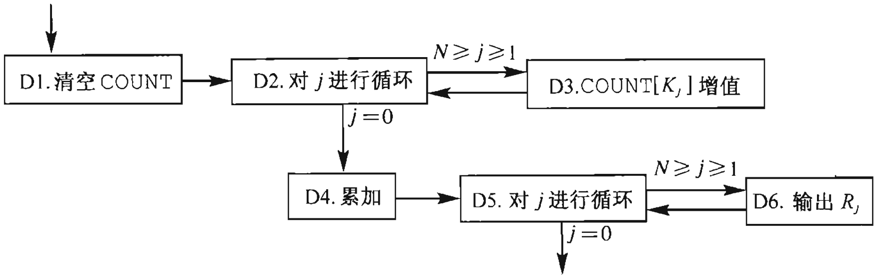


图 9 算法 D：分布计数

这个算法的一个例子在习题 6 中讨论;在习题 9 中有一个 MIX 程序。当范围 $v-u$ 很小时,这个排序过程是非常快的。

如同算法 C 这样通过比较计数进行的排序,首先是由 E. H. Friend 提出的[*JACM* **3**(1956),152],尽管他没有宣称这是他自己的发明。像在算法 D 中那样的分布排序,是由 H. Seward 于 1954 年首先提出的,他把这个方法同后边将讨论的基数排序技术一起使用(见 5.2.5 小节);该方法 W. Feurzeig 也曾以"Mathsort"为名发表过,见 *CACM* **3**(1960),601。

习　题

1.[*15*] 如果在步骤 C2 中,i 从 2 升到 N,而不是从 N 降到 2,算法 C 是否仍然有效? 如果在步骤 C3 中 j 从 1 升到 $i-1$ 又将如何呢?

2.[*21*] 证明当出现相等的键码时,算法 C 仍将正确地工作。如果 $K_j=K_i$ 且 $j<i$,则 R_j 在最后的编序下是在 R_i 之前还是之后出现?

▶3.[*21*] 如果在步骤 C4 中的比较从"$K_i<K_j$"变成为"$K_i\leqslant K_j$",则算法 C 是否仍将正确工作?

4.[*16*] 写出一个 MIX 程序,它"完成"由程序 C 着手的排序;你的程序应把这些键码以递增的次序传送到单元 OUTPUT + 1 至 OUTPUT + N。你的程序需要多少时间?

5.[*22*] 下列一组修改是否改进程序 C?

新添一行 08a:INCX 0,2

改变行 10:JGE 5F

改变行 14:DECX　1

删去行 15。

6.[*18*] 用手算来模仿算法 D,说明当对 16 个记录 5T,0C,5U,0O,9.,1N,8S,2R,6A,4A,1G,5L,6T,6I,7O,7N 进行排序时的中间结果。这里数字是键码,而字母信息附属于这些记录。

7.[*13*] 算法 D 是一个"稳定的"排序方法吗?

8.[*15*] 如果在步骤 D5 中 j 从 1 升到 N,而不是从 N 降到 1,则算法 D 是否仍将正确地工作?

9.[*23*] 为算法 D 写出一个类似程序 C 和习题 4 的 MIX 程序来。作为 N 和$(v-u)$的函数,你的程序的运行时间等于多少?

10.[*25*] 试设计一个有效的算法,当给定 $R_1,\cdots,R_N$ 的值以及$\{1,\cdots,N\}$的排列 $p(1),\cdots,p(N)$后,它分别以$(R_{p(1)},\cdots,R_{p(N)})$代替 N 个量$(R_1,\cdots,R_N)$。试避免使用过多的存储空间(如果希望在地址表排序之后,在存储器中重新安排这些记录,而不要求存储 $2N$ 个记录的空间,就会出现这个问题)。

11.[*M27*] 为习题 10 的算法写出一个 MIX 程序,并分析它的效率。

▶12.[*25*] 试设计一个有效的算法,它适合于在完成表排序(图 7)之后,把记录 $R_1,\cdots,R_N$ 按排好的次序重新排列。试避免使用过多的存储空间。

▶13.[*27*] 算法 D 要求用于存放 $2N$ 个记录 $R_1,\cdots,R_N$ 和 $S_1,\cdots,S_N$ 的空间,证明:如果用新的紧凑的过程来代替步骤 D5 和 D6,则有可能仅需要存放 N 个记录 $R_1,\cdots,R_N$ 的空间(于是这个问题就成为在步骤 D4 之后,根据值 COUNT[u],…,COUNT[v] 设计一个适当地重新排列 $R_1,\cdots,R_N$

的算法，而无须使用附加的存储空间；这实质上是习题 10 中所考虑问题的推广）。

5.2.1 通过插入进行排序

有一类重要的排序技术，是以 5.2 节开头处提到的“玩桥牌者”的方法为基础的。在考察记录 R_j 之前，假定以前的记录 $R_1, \cdots, R_{j-1}$ 已经排好序，然后把 R_j 插入到已排好序的诸记录的适当位置。这个基本主题可以有若干有趣的变形。

直接插入 最简单的插入排序也是最显然的。假定 $1 < j \leqslant N$，而且已经把记录 $R_1, \cdots, R_{j-1}$ 重新排列好，使得（记住：贯穿于本章，K_j 皆表示 R_j 的键码部分）

$$K_1 \leqslant K_2 \leqslant \cdots \leqslant K_{j-1}$$

把新键码 K_j 依次地和 $K_{j-1}, K_{j-2}, \cdots$ 进行比较，直到发现 R_j 应当插入到记录 R_i 和 R_{i+1} 之间为止；然后把记录 $R_{i+1}, \cdots, R_{j-1}$ 向上移动一格，并把新的记录放置到位置 $i+1$ 处。如下列算法所示的那样，宜于把比较和移动操作组合在一起，互相穿插，由于 R_j“被安放到适当的层次中去”，这种排序方法通常称为筛选（*sifting*）或陷入（*sinking*）技术。

算法 S（直接插入排序） 重新安排记录 $R_1, \cdots, R_N$ 到适当位置；在完成排序之后，它们的键码将是有序的，即有 $K_1 \leqslant \cdots \leqslant K_N$。

S1.［对 j 进行循环］ 对于 $j = 2, 3, \cdots, N$ 实施步骤 S2 到 S5；然后终止本算法。

S2.［给 i, K, R 赋值］ 置 $i \leftarrow j-1, K \leftarrow K_j, R \leftarrow R_j$（在下列步骤中，将通过按 i 的递减次序比较 K 和 K_i，来试图把 R 插入正确的位置）。

S3.［比较 $K : K_i$］ 如果 $K \geqslant K_i$，则转向步骤 S5（我们已找到了记录 R 所求的位置）。

S4.［移动 R_i，i 减值］ 置 $R_{i+1} \leftarrow R_i$，然后 $i \leftarrow i-1$。如果 $i > 0$，则返回到步骤 S3（如果 $i = 0$，K 是目前找到的最小的键码，所以记录 R 属于位置 1）。

S5.［R 进入 R_{i+1}］ 置 $R_{i+1} \leftarrow R$。 ▌

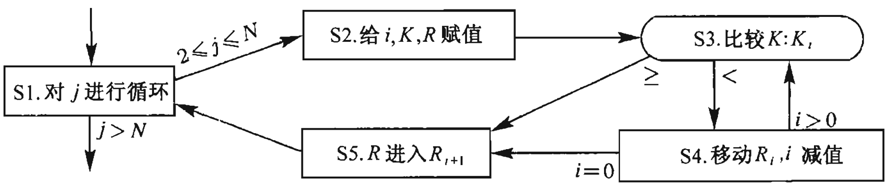


图 10 算法 S：直接插入

表 1 表明，16 个做例子的数，是如何通过算法 S 进行排序的。这个方法在计算机上是很容易实现的；事实上，下列的 MIX 程序是书中最短的，并且还不错的排序程序。

表 1　直接插入的例子

```
 503:087
^
 087  503:512
        ^
 087  503  512:061
^
 061  087  503  512:908
                  ^
 061  087  503  512  908:170
        ^
 061  087  170  503  512  908:897
                       ^
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 061  087  154  170  275  426  503  509  512  612  653  677  765  897  908:703
                                                          ^
 061  087  154  170  275  426  503  509  512  612  653  677  703  765  897 908
```

程序 S(直接插入排序)　有待排序的记录在单元 INPUT + 1 至 INPUT + N 中,它们按一个全字长的键码在同一区域中就地排序。

rI1≡$j-N$;rI2≡i;rA≡R≡K;假定 $N\geq 2$。

START	ENT1	2 - N	1	*S1. 对 j 进行循环, j←2*
2H	LDA	INPUT + N,1	$N-1$	*S2. 对 i,K,R 赋值*
	ENT2	N - 1,1	$N-1$	$i\leftarrow j-1$
3H	CMPA	INPUT,2	$B+N-1-A$	*S3. 比较* $K:K_i$
	JGE	5F	$B+N-1-A$	*如果* $K\geq K_i$,*则转到* S5
4H	LDX	INPUT,2	B	*S4. 移动* R_i,*i 减值*
	STX	INPUT + 1,2	B	$R_{i+1}\leftarrow R_i$
	DEC2	1	B	$i\leftarrow i-1$
	J2P	3B	B	*如果*, $i>0$,*则转到* S3
5H	STA	INPUT + 1,2	$N-1$	*S5. R 进入* R_{i+1}
	INC1	1	$N-1$	
	J1NP	2B	$N-1$	$2\leq j\leq N$ ▌

这个程序的运行时间为 $9B+10N-3A-9$ 个单位,其中 N 是被排序的记录

数,A 是在步骤 S4 中 i 减小到 0 的次数,而 B 是移动的次数。显然,A 是对于 $1<j\leqslant N$, $K_j<(K_1,\cdots,K_{j-1})$的次数;这比自左到右的极小值的个数少 1,所以 A 等价于在 1.2.10 小节中被深入分析的量。略加考虑又使我们看出 B 也是一个熟悉的量:对于固定的 j,移动的次数即是 K_j 的反序的个数。所以,B 是排列 $K_1,K_2,\cdots,K_N$ 的反序的个数。因此,由等式 1.2.10-(16),5.1.1-(12)和 5.1.1-(13),我们有

$$A=(\min 0,\text{ave}H_N-1,\max\ N-1,\text{dev}\sqrt{H_N-H_N^{(2)}});$$

$$B=(\min 0,\text{ave}(N^2-N)/4,\max(N^2-N)/2,\text{dev}\sqrt{N(N-1)(N+2.5)}/6);$$

而且,假定输入键码是不同的并且是随机排列的,则程序 S 的平均运行时间是 $(2.25N^2+7.75N-3H_N-6)u$,习题 33 说明了如何对此稍作改进。

表 1 的例子包含有 16 项;有两个自左至右的极小值的变化,即 087 和 061;而且同上一节我们所见到的那样,有 41 个反序。因此 $N=16$, $A=2$, $B=41$,而总共的排序时间是 514 个时间单位。

二叉插入和两路插入 在一个直接插入排序期间,在处理第 j 个记录时,平均说来要把它的键码大约同$\frac{1}{2}j$ 个此前已排好序的键码进行比较,因此所实施比较的总数大约是$\frac{1}{2}(1+2+\cdots+N)\approx\frac{1}{4}N^2$;当 N 适当大时,这就已经非常之大了。在 6.2.1 小节,将研究"二分查找"技术,该技术使我们能够在仅仅进行 $\lg N$ 次仔细选择的比较之后,就指出在哪里插入第 j 项。例如,当插入第 64 个记录时,可以由对 K_{64}和 K_{32}进行比较开始。如果是小于,则就把它同 K_{16}进行比较,但如果是大于,则就把它同 K_{48}进行比较,等等。于是仅仅做 6 次比较之后,就可知道 R_{64}应插入的位置。插入所有 N 项所做的比较总数就大约是 $N\lg N$,这是对于$\frac{1}{4}N^2$ 的实质性的改进。而 6.2.1 小节表明,其相应的程序并不比直接插入程序复杂许多。这个方法称为二叉插入;它早在 1946 年就由 John Mauchly 在计算机排序的第一个公开讨论中述及。

二叉插入的困难是,它只解决了问题的一半。在已经发现记录 R_j 应插入到哪里之后,仍然需要移动大约$\frac{1}{2}j$ 个此前已排好序的记录,以便为 R_j 腾出位置,所以总共的运行时间实质上仍然同 N^2 成比例。某些早期的计算机(例如 IBM 705)有一个内部的"滚进"指令,它以高速度来进行这样的移动操作,现代的机器通过附加的专用软件,甚至能更快地完成移动操作;但当 N 增加时,对 N^2 的依赖性终将凸现出来。例如,H. Nagler 的分析[*CACM* **3** (1960),618~620]指出,当每个记录的长度为 80 个字符时,在 IBM 705 上对多于约 $N=128$ 个记录的排序不应推荐二叉插入;类似的分析对其它的机器也适用。

当然,一个灵巧的程序员可以想出各种方法来减少所需移动的数量;头一个这样的技巧,如表 2 所示,是早在 50 年代时就被提出的。表中排序的头一项被放置在

一个输出区域的中心,而且通过向右或向左移动(就看哪种最方便)腾出空间。此法比普通二叉插入节省一半运行时间,其代价是程序稍微复杂一点。使用此法时,还可以不必使用比 N 个记录所需要的更多的空间(见习题 6);但对于这个“两路”插入的方法,将不作更详细的叙述,因为已有了更为有趣的方法。

表 2 两路插入

				503			
			∧				
			087	503			
				∧			
			087	503	512		
		∧					
		061	087	503	512		
					∧		
		061	087	503	512	908	
			∧				
	061	087	170	503	512	908	
					∧		
	061	087	170	503	512	897	908
			∧				
061	087	170	275	503	512	897	908

Shell 方法 如果有这样的一个排序算法,它一次只把诸项目移动一个位置,则它的平均运行时间最好也只是同 N^2 成比例。因为在这个排序过程中每个记录都必须平均遍历大约$\frac{1}{3}N$ 个位置(见习题 7)。因此,如果要对直接插入做实质性的改进,就需要一种新原理,它使这些记录做长距离的跳跃,而不只是一些短促的小步挪动。

这样一个方法是由 Donald. L. Shell 于 1959 年提出的[*CACM* 2,7(July 1959),30～32],我们称它为 Shell 排序。表 3 说明了该方法的一般想法:首先把这 16 个记录分成为 8 组,两两一组,即$(R_1,R_9),(R_2,R_{10}),\cdots,(R_8,R_{16})$。分别对每组记录进行排序,使我们进到表 3 的第 2 行,这称为“第一次扫描”。注意,154 已同 512 交换了位置;908 和 897 两者都跳到右边去了。现在把这些记录分成 4 组,每 4 个一组,即$(R_1,R_5,R_9,R_{13}),\cdots,(R_4,R_8,R_{12},R_{16})$,并再次分别对每组进行排序,这“第二次扫描”使我们进到了第 3 行。第三次扫描对于各有 8 个记录的两个组进行排序,然后第四次扫描通过对所有 16 个记录进行排序,来完成整个过程。每个中间排序过程,或者处理比较短的文件,或者处理相当好地编了序的文件,所以每个排序操作可以使用直接插入;这些记录势必快速地收敛到它们的最终目标。

Shell 排序也叫做“减少增量的排序”,因为每一遍通过一个增量 h 来确定,使得我们对相距 h 个单位的记录进行排序。增量 8,4,2,1 的序列不是一成不变的;任何序列 $h_t,h_{t-1},\cdots,h_0$ 都可以使用此方法,只要最后的增量 $h_0=1$ 就行。例如,表 4 示

示出了用增量 7,5,3,1 进行排序的相同数据。有些序列要比其它序列更好些,后面将讨论增量的选择。

表 3 带有增量 8.4.2.1 的 Shell 排序

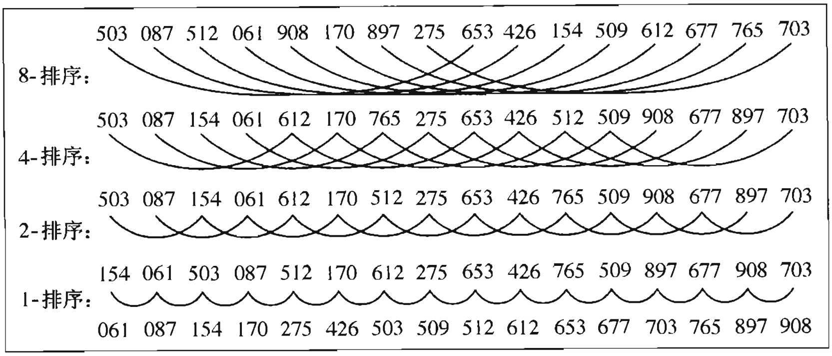
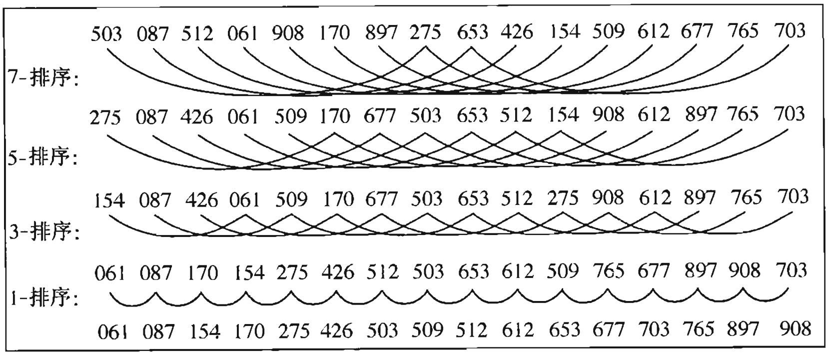


表 4 带有增量 7,5,3,1 的 Shell 排序



算法 D(Shell 排序) 首先适当排列记录 $R_1, \cdots, R_N$;在完成排序之后,它们的键码将是有序的:$K_1 \leqslant \cdots \leqslant K_N$。用一个辅助的增量序列 $h_t, h_{t-1}, \cdots, h_0$ 来控制这个排序过程,其中 $h_0 = 1$;适当选择增量可以显著地减少排序时间,当 $t = 1$ 时,这个算法退化为算法 S。

D1.[对 s 进行循环] 对于 $s = t, t-1, \cdots, 0$ 实施步骤 D2;然后终止这个算法。

D2.[对 j 进行循环] 置 $h \leftarrow h_s$,并对 $h < j \leqslant N$ 实施步骤 D3 到 D6(我们将使用直接插入方法,来对隔开 h 个位置的元素进行排序,使得对于 $1 \leqslant i \leqslant N-h$, $K_i \leqslant K_{i+h}$。步骤 D3 到 D6 实际上分别与算法 S 中的步骤 S2 到 S5 相同)。

D3.[设置 i, K, R] 置 $i \leftarrow j-h$, $K \leftarrow K_j$, $R \leftarrow R_j$。

D4.[比较 $K : K_i$] 如果 $K \geqslant K_i$,则转到步骤 D6。

D5.［移动 R_i，i 减值］ 置 $R_{i+h} \leftarrow R_i$，然后 $i \leftarrow i-h$。如果 $i>0$，则转回步骤 $D4$。

D6.［R 进入 R_{i+h}］ 置 $R_{i+h} \leftarrow R$。 ▌

对应的 MIX 程序不比直接插入程序长很多，下列代码中的 08～19 行是从程序 S 直接翻译过来的，成为算法 D 这个更一般的结构的一部分。

程序 D(Shell 排序) 我们假定这些增量被存储于一个辅助表中，h_s 在单元 $H+s$ 中；所有的增量都小于 N。寄存器分配：$rI1 \equiv j-N$；$rI2 \equiv i$；$rA \equiv R \equiv K$；$rI3 \equiv s$；$rI4 \equiv h$。注意，这个程序是自修改的，为的是要得到内部循环的有效执行。

01	START	ENT3	T-1	1	*D1. 对 s 进行循环。* $s \leftarrow t-1$
02	1H	LD4	H,3	T	*D2. 对 j 进行循环。* $h \leftarrow h_s$
03		ENT1	INPUT,4	T	修改主循环中 3 条指令的地址
04		ST1	5F(0:2)	T	
05		ST1	6F(0:2)	T	
06		ENN1	-N,4	T	$rI1 \leftarrow N-h$
07		ST1	3F(0:2)	T	
08		ENT1	1-N,4	T	$j \leftarrow h+1$
09	2H	LDA	INPUT+N,1	$NT-S$	*D3. 对 i, K, R 赋值*
10	3H	ENT2	N-H,1	$NT-S$	$i \leftarrow j-h$（被修改的指令）
11	4H	CMPA	INPUT,2	$B+NT-S-A$	*D4. 比较 $K:K_i$*
12		JGE	6F	$B+NT-S-A$	如果 $K \geq K_i$，则转到 D6
13		LDX	INPUT,2	B	*D5. 移动 R_i, i 减值*
14	5H	STX	INPUT+H,2	B	$R_{i+h} \leftarrow R_i$（被修改的指令）
15		DEC2	0,4	B	$i \leftarrow i-h$
16		J2P	4B	B	如果 $i>0$，则转到 D4
17	6H	STA	INPUT+H,2	$NT-S$	*D6. R 进入 R_{i+h}*（被修改的指令）
18	7H	INC1	1	$NT-S$	$j \leftarrow j+1$
19		J1NP	2B	$NT-S$	如果 $j \leq N$，则转到 D3
20		DEC3	1	T	

21 J3NN 1B T $t \geqslant s \geqslant 0$ |

***Shell 方法的分析** 为了选择好的用于算法 D 中的增量序列 $h_t, \cdots, h_0$，需要把运行时间作为这些增量的函数来分析。这导致了某些更迷惑人的数学问题，至今犹未完全解决；还没有人能够确定：对于很大的 N 值，最好的增量序列是什么？但是关于 Shell 减少增量排序的特性，已经知道了大量有趣的事实，我们将在这里作一概述。细节在以下的习题中给出（不专长于数学的读者可以跳过下面数页，自公式(12)后的表插入的讨论处继续阅读）。

程序 D 的频率计数指出有 5 个因素确定执行时间：文件大小 N；扫描次数（即增量的个数）$T = t$；增量的和 $S = h_0 + \cdots + h_{t-1}$；比较的次数 $B + NT - S - A$；以及移动的次数 B。如同在程序 S 中分析的那样，A 实质上是在中间的排序操作中遇到的自左到右的极小值的数目，B 是诸子文件中的反序数。对运行时间起支配作用的因素是 B，所以我们将把大部分注意力集中于此。为了进行分析，将假定键码是不同的，且开始时是处于随机的顺序之下。

我们把步骤 D2 的操作称为“h 排序”，于是，Shell 方法由 h_{t-1} 排序，接着是 $h_{t-2}, \cdots$，最后是 h_0 排序所组成。对于 $1 \leqslant i \leqslant N - h$，满足 $K_i \leqslant K_{i+h}$ 的文件，称为“h 有序”的文件。

当恰有两个增量 $h_1 = 2$ 和 $h_0 = 1$ 时，首先考虑直接插入的最简单的推广。在此情况下，第二次扫描开始于一个 2 有序(2-*ordered*)的键码序列 $K_1, K_2, \cdots, K_N$。容易看出，对于 $1 \leqslant i \leqslant n - 2$，使得 $a_i \leqslant a_{i+2}$ 的 $\{1, 2, \cdots, n\}$ 的排列 $a_1 a_2 \cdots a_n$ 的个数是

$$\binom{n}{\lfloor n/2 \rfloor}$$

因为若任选 $\lfloor n/2 \rfloor$ 个元素放入偶数号位置 $a_2 a_4 \cdots$ 中，剩下的 $\lceil n/2 \rceil$ 个元素放入奇数号位置中，则由每次这种选择我们正好得到一个 2 有序排列。在一个随机文件已经 2 排序之后，每种 2 有序排列都是同等可能的。在所有这样的排列当中，反序的平均个数是多少呢？

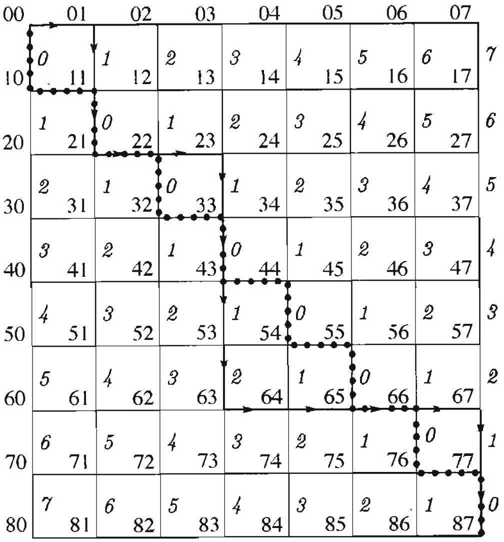


图 11 在 2 有序和一个格子通路之间的对应关系斜体数字是权，它对应于 2 有序排列中的反序数

设 A_n 是在 $\{1, 2, \cdots, n\}$ 的所有 2 有序排列当中的反序总数。显然 $A_1 = 0, A_2 = 1, A_3 = 2$；通过考虑下面 6 种情况

1 3 2 4　1 2 3 4　1 2 4 3　2 1 3 4　2 1 4 3　3 1 4 2

我们发现 $A_4=1+0+1+1+2+3=8$。为了一般地研究 A_n，考虑对于 $n=15$，图 11 所示的"格子框图"。$\{1,2,\cdots,n\}$ 的一个 2 有序排列可以表示为，从左上角的点 $(0,0)$ 到右下角的点 $(\lceil n/2\rceil,\lfloor n/2\rfloor)$ 的一条通路，如果根据在这个排列中，k 出现于奇数或偶数位置而分别作向下或向右的第 k 步通路的话。这个规则确定了 2 有序排列，与格子框图的角到角的 n 步通路之间的一一对应。例如，图 11 中由黑线所示的通路对应于排列

$$2\ 1\ 3\ 4\ 6\ 5\ 7\ 10\ 8\ 11\ 9\ 12\ 14\ 13\ 15 \tag{1}$$

而且，如图 11 所示那样，我们可以把权"附加"到通路的垂直线上；从 (i,j) 到 $(i+1,j)$ 的一条线得到权 $|i-j|$。对格子框图稍做研究就会使读者相信，沿着每条通路，这些权之和等于对应的排列的反序个数；这个和也等于在给定通路和由图中的黑点表示的楼梯通路之间带阴影的方格个数（见习题 12）。例如，(1) 有 $1+0+1+0+1+2+1+0=6$ 个反序。

当 $a\leqslant a'$ 和 $b\leqslant b'$ 时，从 (a,b) 到 (a',b') 的有关通路的条数，是把 $a'-a$ 条垂直线同 $b'-b$ 条水平线混合起来的方式种数，即

$$\binom{a'-a+b'-b}{a'-a}$$

因此，其对应的通路通历从 (i,j) 到 $(i+1,j)$ 的垂直线段的排列数是

$$\binom{i+j}{i}\binom{n-i-j-1}{\lfloor n/2\rfloor-j}$$

乘以相关联的权并对所有线段求和，给出

$$A_{2n}=\sum_{\substack{0\leqslant i\leqslant n\\0\leqslant j\leqslant n}}|i-j|\binom{i+j}{i}\binom{2n-i-j-1}{n-j}$$

$$A_{2n+1}=\sum_{\substack{0\leqslant i\leqslant n\\0\leqslant j\leqslant n}}|i-j|\binom{i+j}{i}\binom{2n-i-j}{n-j} \tag{2}$$

在这些和中的绝对值符号使得此计算有些棘手，但是习题 14 说明 A_n 有惊人的简单形式 $\lfloor n/2\rfloor 2^{n-2}$。因此，在一个随机的 2 有序排列中的平均反序数是

$$\lfloor n/2\rfloor 2^{n-2}\Big/\binom{n}{\lfloor n/2\rfloor}$$

由 Stirling（斯特林）近似公式，这渐近于 $\sqrt{\pi/128}\,n^{3/2}\approx 0.15n^{3/2}$。容易看出反序的极大个数是

$$\binom{\lfloor n/2\rfloor+1}{2}\approx\frac{1}{8}n^2$$

如同在习题 15 中那样，考察生成函数

$$h_1(z)=1$$

$$h_2(z)=1+z$$

$$h_3(z) = 1 + 2z$$
$$h_4(z) = 1 + 3z + z^2 + z^3$$
$$\cdots \qquad (3)$$

以便更加仔细地研究反序的分布，是有教益的。这样一来，我们就发现了标准差也同 $n^{3/2}$ 成比例。所以，这个分布相对于平均值来说不是非常稳定的。

现在来考虑，当增量是 h 和 1 时，算法 D 的一般的两次扫描的情况：

定理 H *在* $\{1,2,\cdots,n\}$ *的一个* h *有序的排列中，反序的平均数是*

$$f(n,h) = \frac{2^{2q-1}q!q!}{(2q+1)!}\left(\binom{h}{2}q(q+1) + \binom{r}{2}(q+1) - \frac{1}{2}\binom{h-r}{2}q\right) \qquad (4)$$

其中，$q = \lfloor n/h \rfloor$，$r = n \bmod h$。

这个定理是 Douglas H. Hunt 给出的[Bachelor's thesis, Princeton University (April 1967)]。注意，当 $h \geqslant n$ 时，这一公式正确地给出 $f(n,h) = \frac{1}{2}\binom{n}{2}$。

证明 一个 h 有序的排列包含长度为 $q+1$ 的 r 个排好序的子序列，和长度为 q 的 $h-r$ 个排好序的子序列。每个反序都来自一对不同的子序列，而且在一个随机的 h 有序的排列中，给定的一对不同的子序列都定义一个随机的 2 有序的排列。因此，反序的平均个数等于在每对不同子序列之间反序的平均个数之和，即

$$\binom{r}{2}\frac{A_{2q+2}}{\binom{2q+2}{q+1}} + r(h-r)\frac{A_{2q+1}}{\binom{2q+1}{q}} + \binom{h-r}{2}\frac{A_{2q}}{\binom{2q}{q}} = f(n,h) \qquad ▌$$

推论 *如果增量序列* $h_{t-1},\cdots,h_1,h_0$ *满足条件*

$$h_{s+1} \bmod h_s = 0 \qquad 对于\ t-1 > s \geqslant 0 \qquad (5)$$

则在算法 D 中移动操作的平均次数是

$$\sum_{t>s\geqslant 1}(r_s f(q_s+1, h_{s+1}/h_s) + (h_s - r_s)f(q_s, h_{s+1}/h_s)) \qquad (6)$$

其中 $r_s = N \bmod h_s$，$q_s = \lfloor N/h_s \rfloor$，$h_t = Nh_{t-1}$，$f$ 在(4)中定义。

证明 h_s 排序的过程由对长度为 $q+1$ 的 r_s 个 (h_{s+1}/h_s) 有序的子文件，以及长度为 q_s 的 $(h_s - r_s)$ 个这样的子文件，进行直接插入排序所组成。整除性条件意味着，在每种 (h_{s+1}/h_s) 有序排列都是同等可能的意义下，每个子文件都是随机的 (h_{s+1}/h_s) 有序的排列，因为我们假定原来的输入是不同元素的一个随机排列。 ▌

当增量分别为 h 和 1 时，在这个推论中的条件(5)对于*两次扫描的* Shell 排序总是满足的。如果 $q = \lfloor N/h \rfloor$ 且 $r = N \bmod h$，则程序 D 中的量 B 的平均值将为

$$\begin{aligned} & rf(q+1,N) + (h-r)f(q,N) + f(N,h) \\ &= \frac{r}{2}\binom{q+1}{2} + \frac{h-r}{2}\binom{q}{2} + f(N,h) \end{aligned}$$

对于第一次近似，函数 $f(n,h)$ 等于 $(\sqrt{\pi}/8)n^{3/2}h^{1/2}$；例如当 $n = 64$ 时，我们可把它同

图 12 的光滑曲线相对照。因此两次扫描的程序 D 的运行时间近似于同 $2N^2/h+\sqrt{\pi N^3 h}$ 成比例。而 h 的最好选择近似于 $\sqrt[3]{16N/\pi}\approx 1.72\sqrt[3]{N}$；而且通过这样选择的 h，我们得到同 $N^{5/3}$ 成比例的平均运行时间。

于是，可以仅仅通过使用具有两个增量的 Shell 方法，就能对直接插入法做出实质性的改进，由 $O(N^2)$ 变成 $O(N^{1.667})$。显然当使用更多的增量时，甚至可以做得更好。习题 18 讨论了当 t 固定且当诸 h 受整除性条件限制时 $h_{t-1},\cdots,h_0$ 的最佳选择；对于很大的 N，运行时间减少到 $O(N^{1.5+\epsilon/2})$，其中 $\epsilon=1/(2^t-1)$。我们不能通过使用上述公式突破 $N^{1.5}$ 的壁垒，因为最后的扫描总是要对这个和提供个反序。

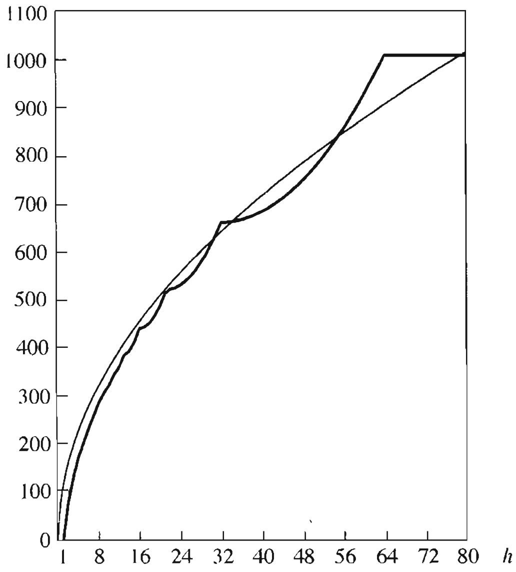


图 12　n 个元素（$n=64$）的 h 有序文件中的反序平均数 $f(n,h)$

$$f(N,h_1)\approx(\sqrt{\pi}/8)N^{3/2}h_1^{1/2}$$

但是凭直觉就知道，当增量 $h_t,\cdots,h_0$ 不满足整除性条件(5)时，甚至能做得更好。例如，8 排序继之以 4 排序，再继之以 2 排序的方法不允许在偶位置和奇位置的键码之间有任何相互作用；因此，最后的 1 排序扫描不可避免地面临 $\theta(N^{3/2})$ 个反序。相反，7 排序继之 5 排序，再继之以 3 排序，以这样的方式交叉掺杂，最后的 1 排序扫描就不会遇到多于 $2N$ 个反序（见习题 26）！确实，这里出现了一个令人吃惊的现象：

定理 K　*如果一个 k 有序的文件被 h 排序，则它保持为 k 有序的。*

于是，首先是 7 排序，然后是 5 排序的一个文件，变成为既是 7 有序的，又是 5 有序的。而如果对它进行 3 排序，则结果是 7,5 和 3 有序的。这个值得注意的性质的例子如表 4 所示。

证明　习题 20 表明，这个定理是下列事实的一个推论：

引理 L　*设 m,n,r 是非负整数，且 $(x_1,\cdots,x_{m+r})$ 和 $(y_1,\cdots,y_{n+r})$ 是任意数列，满足*

$$y_1\leqslant x_{m+1},\ y_2\leqslant x_{m+2},\ \cdots,\ y_r\leqslant x_{m+r} \tag{7}$$

如果诸 x 和诸 y 被独立地排序，使得 $x_1\leqslant\cdots\leqslant x_{m+r}$ 和 $y_1\leqslant\cdots\leqslant y_{n+r}$，则关系(7)仍将成立。

证明　已知除 m 个之外,所有的 x 都高于(即大于或等于)某个 y,这里,不同的 x 高于不同的 y。设 $1\leqslant j\leqslant r$。由于 x_{m+j} 在排序之后高于 $m+j$ 个 x,它至少高于 j 个 y;所以它高于最小的 j 个 y;因此在排序之后 $x_{m+j}\geqslant y_j$。 ▌▌

定理 K 表明,通过互素增量来进行排序是令人满意的,但它并不直接导致对在算法 D 中所做移动次数的精确估计。由于既是 h 有序又是 k 有序的 $\{1,2,\cdots,n\}$ 的排列个数,并不总是 $n!$ 的一个因子,我们可以看到,定理 K 并不提供一切详情;在进行 k 排序和 h 排序之后,某些 k 和 h 有序的文件,比起其它文件更为经常出现。因此,当 $t>3$ 时,对于一般的增量 $h_{t-1},\cdots,h_0$,对于这个算法的平均情况的分析,至今使人们困惑不已。当给定 N 和 $(h_{t-1},\cdots,h_0)$ 时,甚至没有一个明显的方法来求最坏的情况。然而,当增量有某种形式时,关于近似的极大运行时间,我们能够推导出若干个结果出来。

定理 P　*当 $h_s=2^{s+1}-1$,$0\leqslant s<t=\lfloor \lg N\rfloor$ 时,算法 D 的运行时间是 $O(N^{3/2})$。*

证明　只需对于第 s 次扫描中的移动次数 B_s 给出上限,使得 $B_{t-1}+\cdots+B_0=O(N^{3/2})$ 就可以了。对于 $t>s\geqslant t/2$,在前 $t/2$ 次扫描期间,可以使用显然的界限(Bound)$B_s=O(h_s(N/h_s)^2)$;而对于随后的扫描,可以使用习题 23 的结果,$B_s=O(Nh_{s+2}h_{s+1}/h_s)$;因此,$B_{t-1}+\cdots+B_0=O(N(2+2^2+\cdots+2^{t/2}+2^{t/2}+\cdots+2))=O(N^{3/2})$。 ▌

这个定理是 A. A. Papernov 和 G. V. Stasevich 给出的,见 *Problemy Peredachi Informatsii* **1**,3 (1965),81～98。它给出了这个算法的最坏运行时间的上限,而不只是平均运行时间的一个界限。由于当诸 h 满足整除性约定(5)时,最坏运行时间是 N^2 阶的,这个结果是不平凡的;而且习题 24 表明:指数 3/2 已不能再降低了。

1969 年 Vaughan Pratt 发现了定理 P 的一个有趣的改进:*如果被选择的增量是形如 2^p3^q 的所有数的集合,它们都是小于 N 的,则算法 D 的运行时间是 $N(\log N)^2$ 阶的*。在这种情况下,还可以对这个算法做若干重要的简化。参见习题 30 和 31。然而,即使有这些简化,Pratt 的方法由于要对数据做相当多次的扫描,因此仍需要一个很大的开销。所以在实践中,他的增量实际上并不比定理 P 的排序时间更快些,除非 N 是天文数字似地大。对于现实的 N 的最好序列看来要满足 $h_s\approx\rho^s$,其中比例 $\rho\approx h_{s+1}/h_s$ 大体上与 s 无关,但可能依赖于 N。

我们已经注意到,以这样一种方式,即每一个增量都是它的所有前驱者的因子,来选择增量是不明智的。但是,我们将不作出结论说,最好的增量是同它的所有前驱互素的。事实上,当我们是在 g 排序时,既是 gh 排序又是 gk 排序且 $h\perp k$ 的一个文件的每个元素至多有 $\frac{1}{2}(h-1)(k-1)$ 个反序(见习题 21)。通过分析这个事实,当 $N\to\infty$ 时 Pratt 的序列 $\{2^p3^q\}$ 获胜,但是对于实用的目的说来它提高得太小。

Janet Incerpi 和 Robert Sedgewick[*J. Comp. Syst. Sci.* **31** (1985),210～224;又见 *Lecture Notes in Comp. Sci.* **1136** (1996),1～11]通过说明如何构造满足 $h_s\approx\rho^s$,而

且每个增量还是它的两个前驱者的最高公因子，找到理论和实践两方面都最好的增量序列。给定任何数 $\rho>1$，他们从定义一个基序列 $a_1,a_2,\cdots$ 开始(其中 a_k 为 $\geqslant\rho^k$ 的最小整数)，使得对于 $1\leqslant j<k$，$a_j\perp a_k$。例如，如果 $\rho=2.5$，这个基序列为

$$a_1,a_2,a_3,\cdots = 3,7,16,41,101,247,613,1529,3821,9539,\cdots$$

现在通过置 $h_0=1$ 和对于 $\binom{r}{2}<s\leqslant\binom{r+1}{2}$，置

$$h_s = h_{s-r}a_r \tag{8}$$

定义增量。因此增量序列开始于

$$1;\ a_1;\ a_2,\ a_1a_2;a_1a_3,\ a_2a_3,\ a_1a_2a_3;\ \cdots$$

例如，当 $\rho=2.5$ 时，我们得到

1,3,7,21,48,112,336,861,1968,4592,13776,33936,86861,198768,…

关键点是我们可以把递推式(8)转过来

$$h_s = h_{r+s}/a_r = h_{\binom{r}{2}}/a_{\binom{r}{2}-s} \qquad 对于\binom{r-1}{2}\leqslant s\leqslant\binom{r}{2} \tag{9}$$

因此，由上一段中的论证，当我们进行 h_0-排序，h_1-排序…时，每个元素的反序个数至多为

$$b(a_2,a_1);b(a_3,a_2),b(a_3,a_1);b(a_4,a_3),b(a_4,a_2),b(a_4,a_1);\cdots \tag{10}$$

其中 $b(h,k)=\frac{1}{2}(h-1)(k-1)$。如果 $\rho^{t-1}\leqslant N<\rho^t$，移动的总数 B 至多是 N 乘以这个序列的头 t 个元素之和。因此(参见习题 41)，我们可以证明，最坏情况的运行时间比 $N^{1.5}$ 的阶要好得多。

定理 I *当增量 h_s 由(8)定义时，算法 D 的运行时间为 $O(Ne^{c\sqrt{\ln n}})$。这里 $c=\sqrt{8\ln\rho}$ 并且由 O 所隐含的常数依赖于 ρ。* ▌

当 $N\to\infty$ 时这个渐近上限并不特别重要，因为 Pratt 的序列就做得很好。定理 I 主要是具有实际增长率 $h_s\approx\rho^s$ 的一个增量序列，当给定任何 $\rho>1$ 的值时，可以保证对于任意小的 $\epsilon>0$，是 $O(N^{1+\epsilon})$ 的。

让我们通过考察程序 D 的总共的运行时间，即 $(9B+10NT+13T-10S-3A+1)u$，来更仔细地考虑 N 的实际大小。表 5 所示为当 $N=8$ 时，对于各种增量序列的平均运行时间。对于这个小的 N 值，簿记操作是费用的最重要部分。因此当 $t=1$ 时得到最好的结果；因此，对于 $N=8$，使用简单的直接插入更好(当 $N=8$ 时程序 S 的平均运行时间仅为 $191.85u$)。奇怪的是，当 $h_1=6$ 时，出现最好的两遍扫描算法，因为在这里大的 S 值要比小的 B 值更为重要。类似地，三个增量 3 2 1 极小化平均移动次数，但它们并不导致最好的三次扫描序列。在这里来记录极大化移动次数的某些排列的“最坏情况”可能是有趣的，因为对于这样的排列的一般结构仍然是未知的：

$h_2=5,h_1=3,h_0=1$: 85263741 (19 次移动)

$h_2=3,h_1=2,h_0=1$: 83572461 (17 次移动)

表 5 当 $N=8$ 时,对算法 D 的分析

增量	A_{ave}	B_{ave}	S	T	MIX 时间
1	1.718	14.000	1	1	204.85u
2 1	2.667	9.657	3	2	235.91u
3 1	2.917	9.100	4	2	220.15u
4 1	3.083	10.000	5	2	217.75u
5 1	2.601	10.000	6	2	209.20u
6 1	2.135	10.667	7	2	206.60u
7 1	1.718	12.000	8	2	209.85u
4 2 1	3.500	8.324	7	3	274.42u
5 3 1	3.301	8.167	9	3	253.60u
3 2 1	3.320	7.829	6	3	280.50u

当 N 增长得更大时,我们有一个稍微不同的情况。表 6 所示为当 $N=1000$ 时,各种增量序列的近似移动次数,最初的几项满足整除性限制(5),使得可以使用公式(6)和习题 19;通过使用经验检验(Empirical tests)可以得到其他情况的近似平均值。生成了 1000 个元素的一万个随机文件,而且它们每一个都用每个增量序列加以排序。自左至右的极小 A 的数的标准差通常约为 15;移动次数 B 的标准差通常约为 300。

在这些数据中某些模式是明显的,但算法 D 的行为仍然是非常模糊的。Shell 最初建议使用增量$\lfloor N/2\rfloor$,$\lfloor N/4\rfloor$,$\lfloor N/8\rfloor$,…,但当 N 的表示包含一长串 0 时,这是不合适的。Lazarus 和 Frank[*CACM* **3** (1960),20～22]建议使用实质上相同的序列,但必要时加 1,以使所有增量都成为奇数。Hibbard[*CACM* **6** (1963),206～213]建议使用形如 2^k-1 的增量;Papernov 和 Stasevich 建议使用形如 2^k+1 的增量。表 6 中研究的其它自然序列包含数$(2^k-(-1)^k)/3$、$(3^k-1)/2$ 以及斐波那契数,和对于 $\rho=2.5$ 和 $\rho=2$ 的 Incerpi-Sedgewick 序列(8),另外还示出类似 Pratt 的序列 $\{5^p11^q\}$ 和 $\{7^p13^q\}$,因为它们保留 $O(N(\log N)^2)$ 的渐近特性,但对于小的 N 有较低的开销费用。表 6 中最后的一些例子来自于由 Sedgewick 设计的另外的序列,它们基于稍微不同的带启发式的探索[*J. Algorithms*. **7** (1986),159～173]:

$$h_s=\begin{cases}9.2^s-9.2^{s/2}+1 & \text{如果 } s \text{ 为偶数}\\ 8.2^s-6.2^{(s+1)/2}+1 & \text{如果 } s \text{ 为奇数}\end{cases} \tag{11}$$

当使用这些增量$(h_0,h_1,h_2,\cdots)=(1,5,19,41,109,209,\cdots)$时,Sedgewick 证明,最坏情况的运行时间是 $O(N^{4/3})$。

对于形如 2^k+1 的增量,以及对于 $\rho=2$ 的 Incerpi-Sedgewick 序列,观察到极小

的移动次数大约是 6750。但重要的是要认识到，移动次数并非惟一的考虑，即使是它支配渐近运行时间。因为程序 D 花费 $9B+10(NT-S)+\cdots$ 个时间单位，我们看到一次扫描大约相当于减少 $\frac{10}{9}N$ 次移动；当 $N=1000$ 时，如果我们能减少一次扫描，我们宁愿增加 1111 次移动。（无论如何，如果 h_{t-1} 接近 N，第一次扫描是非常快的，因为 $NT-S=(N-h_{t-1})+\cdots+(N-h_0)$。）

表 6　当 $N=1000$ 时，算法 D 的近似特性

											增量		A_{ave}	B_{ave}	T
												1	6	249750	1
											17	1	65	41667	2
										60	6	1	158	26361	3
									140	20	4	1	262	21913	4
								256	64	16	4	1	362	20459	5
							576	192	48	16	4	1	419	20088	6
						729	243	81	27	9	3	1	378	18533	7
			512	256	128	64	32	16	8	4	2	1	493	16435	10
				500	250	125	62	31	15	7	3	1	516	7655	9
				501	251	125	63	31	15	7	3	1	558	7370	9
				511	255	127	63	31	15	7	3	1	559	7200	9
					255	127	63	31	15	7	3	1	436	7445	8
						127	63	31	15	7	3	1	299	8170	7
							63	31	15	7	3	1	190	9860	6
								31	15	7	3	1	114	13615	5
			513	257	129	65	33	17	9	5	3	1	561	6745	10
				257	129	65	33	17	9	5	3	1	440	6995	9
					129	65	33	17	9	5	3	1	304	7700	8
						65	33	17	9	5	3	1	197	9300	7
							33	17	9	5	3	1	122	12695	6
			683	341	171	85	43	21	11	5	3	1	511	7365	10
				341	171	85	43	21	11	5	3	1	490	7490	9
							255	63	15	7	3	1	373	8620	6
							257	65	17	5	3	1	375	8990	6
							341	85	21	5	3	1	410	9345	6
377	233	144	89	55	34	21	13	8	5	3	2	1	518	7400	13
	233	144	89	55	34	21	13	8	5	3	2	1	432	7610	12
						377	144	55	21	8	3	1	456	8795	7
						365	122	41	14	5	2	1	440	8085	7
							364	121	40	13	4	1	437	8900	6

（续）

增量									A_{ave}	B_{ave}	T
				121	40	13	4	1	268	9790	5
		336	112	48	21	7	3	1	432	7840	7
306	170	90	45	18	10	5	2	1	465	6755	9
			169	91	49	13	7	1	349	8698	6
	275	125	121	55	25	11	5	1	446	6788	8
		190	84	37	16	7	3	1	359	7201	7
	929	505	209	109	41	19	5	1	512	7725	8
		505	209	109	41	19	5	1	519	7790	7
			209	109	41	19	5	1	382	8165	6

M.A.Weiss[*Comp. J.* **34** (1991),88～91]所进行的广泛的实验测试明显地提示，对于增量 $2^k-1,\cdots15,7,3,1$ 由算法 D 所实现的平均移动次数近似地同 $N^{5/4}$ 成正比。更精确地说，Weiss 发现，当使用这些增量时，对于 $100\leqslant N\leqslant12000000$，$B_{ave}\approx 1.55\ N^{5/4}-4.48N+O/N^{3/4}$；实验的标准差近似为 $.065N^{5/4}$。另一方面，Marcin Ciura 的子序列测试表明，Sedgewick 的序列(11)显然有 $B_{ave}=O(N(\log N)^2)$ 或者更好。当 $N\leqslant10^6$ 时，序列(11)的标准差惊人地小，但当 N 超过 10^7 时，它不可思议地爆发性地增长。

利用形如 $2^k-1,2^k+1$ 和(11)的增量，表 7 示出了 3 次随机实验中所得到的，每次扫描的移动次数的统计分析。在每种情况下，使用相同的数的文件。在 3 种情况下，总共的移动次数 $\sum_s B_s$ 达到 346152，329532，248788，所以在这个例子中序列(11)显然是优越的。

表 7　每遍扫描的移动：对于 $N=20000$ 时的实验

h_s	B_s	h_s	B_s	h_s	B_s
4095	19458	4097	19459	3905	20714
2047	15201	2049	14852	2161	13428
1023	16363	1025	15966	929	18206
511	18867	513	18434	505	16444
255	23232	257	22746	209	21405
127	28034	129	27595	109	19605
63	33606	65	34528	41	26604
31	40350	33	45497	19	23441
15	66037	17	48717	5	38941
7	43915	9	38560	1	50000
3	24191	5	20271		
1	16898	3	9448		
		1	13459		

尽管逐渐地人们更好地理解了算法 D,但是超过 30 年的研究并没有能找出任何充分的根据,以得出关于什么增量序列能使它最有效的强有力的断言。如果 N 小于 1000,如下的一个简单规则

$$令\ h_0 = 1, h_{s+1} = 3h_s + 1, 且当\ h_{t+1} > N\ 时,以\ h_{t-1}\ 停止 \tag{12}$$

似乎与任何其它增量一样好。对于更大的 N 值,可以推荐 Sedgewick 的序列(11)。然而,N. Tokuda 使用$\lfloor 2.25h_s \rfloor$代替式(12)中的 $3h_s$,得到更好的结果,甚至可能有 $N\log N$ 的阶。请见*Information Processing 92* **1**(1992),449~457。

表插入 现在我们放下 Shell 方法,来考虑有关直接插入的其它类型的改进。对一个给定算法进行改进的最重要的一般方式之一是,仔细地检查它的数据结构,因为,为避免不必要的操作而进行的数据结构的再组织,常常取得实质性的改进。2.4 节中有这个一般想法的进一步讨论,并研究了一个颇为复杂的算法;我们考虑如何把它应用于像直接插入这样非常简单的算法。对于算法 S,什么是最适当的数据结构呢?

直接插入包括两个基本的操作:

i) 扫描一个有序文件,找出小于或等于一个给定键码的最大键码;

ii) 插入一个新的记录到一个有序文件的指定部分。

这个文件显然是一个线性表,而算法 S 通过使用顺序分配(见 2.2.2 小节)来处理这个表;因此,为完成每一个插入操作,必须移动大约一半记录。另一方面,我们知道,链接分配(见 2.2.3 小节)对插入比较理想,因为仅仅需要改变较少的链接;而且另一个操作,即顺序扫描,对链接分配也大致像顺序分配一样容易进行。由于总是从同一方向来扫描这张表,故只需要单路链接。因此得出结论,对于直接插入,“适用的”数据结构是单路链接的线性表。不难修正算法 S,以便按递增的次序来扫描这个表:

算法 L(*表插入*) 假定记录 $R_1, \cdots, R_N$ 包含键码 $K_1, \cdots, K_N$,且链接字段 $L_1, \cdots, L_N$能容纳数字 0 到 N;在这个文件的开始处,一个人为的记录 R_0 中有另一个链接字段 L_0。本算法设置链接字段,使得这些记录以递增的顺序链接在一起。于是,如果 $p(1)\cdots p(N)$是使得 $K_{p(1)} \leqslant \cdots \leqslant K_{p(N)}$的稳定的排列,则这个算法将得出

$$L_0 = p(1); \quad L_{p(i)} = p(i+1), \quad 对于\ 1 \leqslant i \leqslant N; \quad L_{p(N)} = 0 \tag{13}$$

L1. [对 j 进行循环] 置 $L_0 \leftarrow N, L_N \leftarrow 0$($L_0$ 起表“头”的作用,0 起空链的作用;因此这个表实质上是循环的)。对于 $j = N-1, N-2, \cdots, 1$,执行步骤 L2 到 L5;然后终止本算法。

L2. [对 p, q, K 赋值] 置 $p \leftarrow L_0, q \leftarrow 0, K \leftarrow K_j$(在下列步骤中,通过把 K 按递增次序同以前的键码进行比较,来把 R_j 插入到链接表中的适当位置。变量 p 和 q 起指向这个表的当前位置的指针作用,而且 $p = L_q$,所以 q 比 p 落后一步)。

L3.［比较 $K:K_p$］ 如果 $K \leqslant K_p$ 则转向步骤 L5(已经在这个表的 R_q 和 R_p 之间,为记录 R 找到了所要求的位置)。

L4.［碰撞 p,q］ 置 $q \leftarrow p, p \leftarrow L_q$。如果 $p>0$,则转回到步骤 L3(如果 $p=0$,则 K 是当前发现的最大的键码;因此记录 R 列入这个表的末端,在 R_q 和 R_0 之间)。

L5.［插入表中］ 置 $L_q \leftarrow j, L_j \leftarrow p$。 ▌

这个算法是重要的,不仅仅因为它是简单的排序方法,而且还因为它经常作为其它表处理算法的一部分出现。表 8 示出了当对 16 个数进行排序时开头几步的情况。习题 32 给出最后的链接设置。

表 8 表插入的例子

j:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
K_j:	—	503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703
L_j:	16	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	0
L_j:	16	—	—	—	—	—	—	—	—	—	—	—	—	—	—	0	15
L_j:	14	—	—	—	—	—	—	—	—	—	—	—	—	—	16	0	15
……																	

程序 L(*表插入*) 我们假定 K_j 存储于 INPUT + j(0:3)中,L_j 存储于 INPUT + j(4:5)中。rI1≡j,rI2≡p;rI3≡q;rA(0:3)≡K。

```
KEY     EQU   0:3
LINK    EQU   4:5
START   ENT1  N               1           L1.对j进行循环。j←N
        ST1   INPUT(LINK)     1           L0←N
        STZ   INPUT+N(LINK)   1           LN←0
        JMP   6F              1           转去使j减值
2H      LD2   INPUT(LINK)     N-1         L2.对p,q,K赋值。p←L0
        ENT3  0               N-1         q←0
        LDA   INPUT,1         N-1         K←Kj
3H      CMPA  INPUT,2(KEY)    B+N-1-A     L3.比较K:Kp
        JLE   5F              B+N-1-A     如果K≤Kp,则转L5
4H      ENT3  0,2             B           L4.碰撞p,q。q←p
        LD2   INPUT,3(LINK)   B           p←Lq
        J2P   3B              B           如果p>0,则转L3
```

```
5H    ST1     INPUT,3(LINK)    N-1      L5.插入表中。Lq←j
      ST2     INPUT,1(LINK)    N-1      Lj←p
6H    DEC1    1                N
      J1P     2B               N        N>j≥1  ▌
```

这个程序的运行时间为 $7B+14N-3A-6$ 个单位，其中 N 是文件的长度，$A+1$ 是自右到左极大值的个数，B 是原来排列中反序的个数（参考程序 S 的分析。注意程序 L 不重新排列内存中的这些记录；这可以如习题 5.2-12 中那样，以大约 $20N$ 的额外时间单位完成之）。程序 S 需要 $(9B+10N-3A-9)u$ 时间单位，而且由于 B 约为 $\frac{1}{4}N^2$，我们可以看出，用作链接字段的额外存储空间已经使我们节省了大约 22% 的执行时间。通过周密的程序设计，还可以再节省 22%（见习题 33），但运行时间仍保持同 N^2 成比例。

下面对至今所做的工作做一总结　我们开始于一个简单而且自然的排序算法 S，它进行了大约 $\frac{1}{4}N^2$ 次比较和 $\frac{1}{4}N^2$ 次移动。一方面，通过考虑二叉插入来进行改进，后者大约进行 $N\lg N$ 次比较和 $\frac{1}{4}N^2$ 次移动。通过“两路插入”稍稍改变数据结构，使移动次数减少到大约 $\frac{1}{8}N^2$。Shell 的减少增量排序技术对于在实用范围中的 N，使比较和移动次数减少到大约 $N^{7/6}$；当 $N\to\infty$ 时，这个数可以被减少到阶为 $N(\log N)^2$。对算法 S 进行改进的另一条途径是使用一个链接的数据结构，结果就得到表插入方法，它大约进行 $\frac{1}{4}N^2$ 次比较，0 次移动和 $2N$ 次改变链接。

能不能把这些方法的最好特性结合起来，使之既像二叉插入那样把比较次数减少到 $N\log N$ 的阶数，又像表插入那样减少移动的次数呢？答案是肯定的，方法是采取一个树结构的排列。这个可能性首先是由 D. J. Wheeler 于 1957 年阐明的；他建议使用两路插入，直到有必要移动某些数据时为止；然后，不去移动这些数据，而是插入指向另一内存区域的指针，而且同样的技术又递归地应用到所有有待插入到这个新内存区域去的项目上。Wheeler 最初的方法［见 A. S. Douglas, *Comp. J.* **2** (1959), 5］是顺序内存和链接内存的一个复杂组合，且带有可变大小的节点；对于我们的 16 个例数，将形成图 13 的树。利用二叉树，一个类似的但较为简单的树插入方案由 C. M. Berners-Lee，大约于 1958 年独立地设计出来［见 *Comp. J.* **3** (1960), 174, 184］。由于此方法以及它的改进，对于查找和排序都十分重要，所以我们将在第六章详细地讨论它们。

另外，还有一条改进直接插入法的途径，即考虑一次插入若干个项目。例如，假设我们有一个包含 1000 个项目的文件，如果它们中的 998 个项目已经排好序，那么算法 S 还要再对整个文件进行两次扫描（首先插入 R_{999}，然后插入 R_{1000}）。如果我

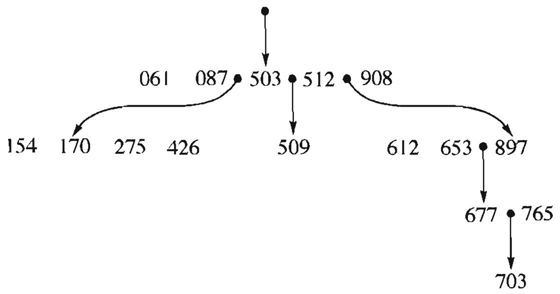


图 13 Wheeler 树插入方案示例

们首先比较 R_{999} 和 R_{1000}，看哪个更大，然后通过对这个文件的一次考察来插入它们俩，则很明显可以节省时间。这种类型的组合操作，大约包含 $\frac{2}{3}N$ 次比较和移动(参见习题 3.4.2-5)，而不是每次大约包含 $\frac{1}{2}N$ 个比较和移动的两次扫描。

换句话说，一般说来，对于要求长时间的查找操作，进行"分批处理"是一个好的想法，使得多个操作可以一起完成。如果把这个想法引向其自然的结论，就会重新发现通过合并进行排序的方法。它如此重要，因此将在 5.2.4 小节中进行讨论。

地址计算排序 至此，有关简单的直接插入方法，已详细地讨论了所有可能的改进途径；但是让我们再考虑一下下面的情况！假设你要按照作者名字的次序把几十本书放到书架上，而这些书是以随机的顺序给你的。当你放书时，你自然尝试来估计每本书的最后位置，由此来减少比较的次数和你要做的移动。而且如果你通过比需要的绝对空间稍稍多一点的书架空间开始，整个进程将会稍微更有效些。

这个方法首先由 Isaac 和 Singleton 提出来用于计算机的排序(见 *JACM* **3** (1956)，169～174)，而由 Tarter 和 Kronmal 进一步加以发展(见 *Proc. ACM Nat'l Conf.* **21** (1966)，331～337)。

地址计算排序通常要求同 N 成比例的附加存储空间，或者用来保留足够的空间使得不需要做过多的移动，或者用来保留辅助的表格，用这些表格帮助处理键码分布的不规则性(见"分布计数"排序，算法 5.2D，它是地址计算的一种形式)。如果如同在表插入方法中那样，把这个附加的存储空间提供给链接字段，那么，也许可以最好地使用它们。由此，还可以避免为输入和输出开设独立的区域；一切都能够在同一个存储区域中完成。

根据这些考虑，我们主要推广表插入，但这里保持的是若干张表，而不仅仅是一张。每张表被用于键码的某些范围。我们做重要的假定，即假定这些键码都是相当均匀地分布的，而不是不规则地"聚集起来的"：把键码的所有可能值的集合划分成 M 部分，并假定一个给定的键码落入一个给定部分的概率是 $1/M$。然后，提供 M 个表头的附加存储，而且每一个表就像在简单的表插入中那样来处理。

这里不必详细地给出算法；这个方法只是简单地以把所有表头置为 Λ 开始。当

每个新项目进入时，首先决定它的键码落入 M 个部分中的哪一个，然后就像在算法 L 中那样把它插入到对应的表中。

为了说明这个方法，假设前面示例中的 16 个键码被分成 $M=4$ 个范围 0～249，250～499，500～749，750～999。随着键码 $K_1, K_2, \cdots, K_{16}$ 逐个地被插入，我们得到下列的配置：

	4 个项目之后	8 个项目之后	12 个项目之后	最后的状态
表 1：	061,087	061,087,170	061,087,154,170	061,087,154,170
表 2：		275	275,426	275,426
表 3：	503,512	503,512	503,509,512,653	503, 509, 512, 612, 653, 677,703
表 4：		897,908	897,908	765,897,908

（以下的程序 M 实际上是以相反的次序，即 $K_{16}, \cdots, K_2, K_1$，来插入键码的，但最后的结果相同）。由于使用了链接存储，所以可变长度的表不引起存储分配的问题。如果愿意，所有的表都可在最后组成一个表（参见习题 35）。

程序 M（多表插入）　在本程序中，做像在程序 L 中一样的假定，但是这些键码必须是非负的，于是

$$0 \leqslant K_j < (\text{BYTESIZE})^3$$

通过把每个键码乘以一个适当的常数，这个程序把上述范围分为 M 个相等的部分。表头在单元 HEAD + 1 到 HEAD + M 中。

```
KEY     EQU   1:3
LINK    EQU   4:5
START   ENT2  M                  1    
        STZ   HEAD,2             M    HEAD[p]←Λ
        DEC2  1                  M    
        J2P   *-2                M    M≥p≥1
        ENT1  N                  1    j←N
2H      LDA   INPUT,1(KEY)       N    
        MUL   =M(1:3)=           N    rA←⌊M·K_j/BYTESIZE^3⌋
        STA   *+1(1:2)           N    
        ENT4  0                  N    rI4←rA
        INC3  HEAD+1-INPUT,4     N    q←LOC(HEAD[rA])
        LDA   INPUT,1            N    K←K_j
        JMP   4F                 N    转去对 p 赋值
```

```
15  3H   CMPA  INPUT,2(KEY)    B+N-A
16       JLE   5F              B+N-A   若 K≤K_p,则去插入
17       ENT3  0,2             B       q←p
18  4H   LD2   INPUT,3(LINK)   B+N     p←LINK(q)
19       J2P   3B              B+N     如果不是表的末端则转移
20  5H   ST1   INPUT,3(LINK)   N       LINK(q)←LOC(R_j)
21       ST2   INPUT,1(LINK)   N       LINK(LOC(R_j))←p
22  6H   DEC1  1               N
23       J1P   2B              N       N≥j≥1  ▌
```

这个程序是对一般的 M 写的，但是，倒不如把 M 固定在某个合适的值上；例如，可以选择 M = BYTESIZE，使得这些表头都能通过一条 MOVE 指令来清零，并用一条 LD3 INPUT,1(1:1)指令即可代替 08 行～11 行的乘法序列。在程序 L 和程序 M 之间最值得注意的差别在于这样一个事实，即当不需进行比较时，程序 M 必须考虑一个空表的情况。

用 M 个表节省了多少时间呢？程序 M 的总运行时间为 $7B+31N-3A+4M+2$ 个单位，其中 M 为表的个数，N 为被排序记录的个数，而 A 和 B 分别计算自右到左的极大值个数和每个表键码当中的反序个数（同本节的其它时间分析相反，在这里把一个非空排列的最右元素包括在计数 A 中）。我们已经对于 $M=1$ 研究了 A 和 B，当时它们的平均值分别是 H_N 和 $\left(\frac{1}{2}\right)\binom{N}{2}$。按关于键码分布的假定，一个给定的表在排序结束时恰包含 n 个项目的概率是“二等式”概率

$$\binom{N}{n}\left(\frac{1}{M}\right)^n\left(1-\frac{1}{M}\right)^{N-n} \tag{14}$$

因此在一般情况下 A 和 B 的平均值是

$$A_{\text{ave}} = M\sum_n \binom{N}{n}\left(\frac{1}{M}\right)^n\left(1-\frac{1}{M}\right)^{N-n} H_n \tag{15}$$

$$B_{\text{ave}} = M\sum_n \binom{N}{n}\left(\frac{1}{M}\right)^n\left(1-\frac{1}{M}\right)^{N-n} \binom{n}{2}\Big/2 \tag{16}$$

利用等式 1.2.6-(20)的一个特殊情况的恒等式

$$\binom{N}{n}\binom{n}{2} = \binom{N}{2}\binom{N-2}{n-2}$$

我们可以很容易地计算(16)中的和：

$$B_{\text{ave}} = \frac{1}{2M}\binom{N}{2} \tag{17}$$

而且习题 37 导出了 B 的标准差。但(15)中的和是更困难的，由定理 1.2.7A，我们有

$$\sum_{n}\binom{N}{n}(M-1)^{-n}H_n=\left(1-\frac{1}{M}\right)^{-N}(H_N-\ln M)+\epsilon$$

$$0<\epsilon=\sum_{n>N}\frac{1}{n}\left(1-\frac{1}{M}\right)^{n-N}<\frac{M-1}{N+1}$$

因此

$$A_{\text{ave}}=M(H_N-\ln M)+\delta,\quad 0<\delta<\frac{M^2}{N+1}\left(1-\frac{1}{M}\right)^{N+1} \tag{18}$$

(当 $M\approx N$ 时这个公式实际上是没有用的;习题 40 给出当 $M=N/\alpha$ 时,对于 A 的渐近特性更详细的分析)。

通过组合(17)和(18),我们可以导出当 $N\to\infty$ 时对于固定的 M,程序 M 的总的运行时间为:

$$\begin{aligned}&\min && 31N+M+2\\&\text{ave} && 1.75N^2/M+31N-3MH_N+3M\ln M+4M-3\delta-1.75N/M+2\\&\max && 3.50N^2+24.5N+4M+2\end{aligned} \tag{19}$$

注意,当 M 不太大时,*我们把平均时间加快 M 倍*;$M=10$ 大约要比 $M=1$ 快 10 倍!然而,极长时间比平均时间要大得多;这重申了关于键码分布相当均等的假定,因为当所有记录都堆积到同一个表时,出现了最坏的情况。

如果置 $M=N$,则平均运行时间近似于 $34.36N$;当 $M=\frac{1}{2}N$ 时,它稍大些,近似于 $34.52N$;而当 $M=\frac{1}{10}N$ 时,它近似于 $48.04N$。习题 35 中的补充程序,把所有的 M 个表链接成一个表,其额外的费用分别把这些时间升高到 $44.99N$,$41.95N$ 和 $52.74N$(注意这些 MIX 时间单位中,有 $10N$ 个是花费在乘法指令中的)。*仅仅假定这些键码相当好地散布在它们的范围中,我们就实现了一个阶为 N 的排序方法。*

在 5.2.5 小节讨论了对于多个表插入的一些改进。

习　题

1. [*10*] 算法 S 是一个稳定的排序算法吗?

2. [*11*] 如果把步骤 S3 中的关系“$K\geqslant K_i$”改为“$K>K_i$”,则算法 S 是否仍将正确地排序?

▶3. [*30*] 程序 S 是不是用 MIX 所能编写的最短的排序程序,换句话说,是否有实现同样效果的一个更短的程序?

▶4. [*M20*] 作为 N 的一个函数,求出程序 S 的极小和极大运行时间。

▶5. [*M27*] 给定$\{1,2,\cdots,N\}$的一个随机排列作为输入,求程序 S 总运行时间的生成函数 $g_N(z)=\sum_{k\geqslant 0}p_{Nk}z^k$,其中 p_{Nk} 是程序 S 恰好花 k 个单位时间的概率。此外对给定的 N,计算运行时间的标准差。

6. [*23*] 表 2 说明的两路插入方法,似乎意味着除了有一个能容纳 N 个记录的输入区域之

外,还有能容纳达 $2N+1$ 个记录的一个输出区域。试证说明两路插入可以仅仅使用足以存放 $N+1$ 个记录的空间来完成,同时包括输入和输出在内。

7. [*M20*] 如果 $a_1a_2\cdots a_n$ 是 $\{1,2,\cdots,n\}$ 的一个随机排列,问什么是 $|a_1-1|+|a_2-2|+\cdots+|a_n-n|$ 的平均值(此为 n 乘上在一个排序过程中某记录所走过的平均纯距离)?

8. [*10*] 算法 D 是一个稳定的排序算法吗?

9. [*20*] 对应于表 3 和表 4,量 A 和 B 是多少? 程序 D 的总运行时间是多少? 试讨论在这种情况下 Shell 方法相对于直接插入的优点。

▶10. [*22*] 如果当步骤 D3 开始时 $K_j \geqslant K_{j-h}$,则算法 D 指定了许多什么贡献也没有的动作。说明怎样修改程序 D,使得可以避免这些多余的计算,并讨论这样一个修改的优点。

11. [*M10*] 在如图 11 所示的一个格中,对应于排列 1 2 5 3 7 4 8 6 9 11 10 12 的通路是什么?

12. [*M20*] 证明在一个格通路和楼梯通路之间的区域(见图 11),等于对应 2 有序(2 - ordered)排列中的反序的个数。

▶13. [*M16*] 说明怎样把权放置到一个格的水平线段上,而不是垂直线段上,使得在格的一条通路上水平的权之和,是对应的 2 有序排列中反序的数目。

14. [*M28*] (a) 说明在由等式(2)所定义的和中,$A_{2n+1}=2A_{2n}$。(b) 如果我们置 $r=s, t=-2$,习题 1.2.6 - 26 的一般恒等式简化为

$$\sum_k \binom{2k+s}{k} z^k = \frac{1}{\sqrt{1-4z}}\left(\frac{1-\sqrt{1-4z}}{2z}\right)^s$$

通过考虑和式 $\sum_n A_{2n}z^n$,证明

$$A_{2n} = n \cdot 4^{n-1}$$

▶15. [*HM33*] 设 $g_n(z)$、$\hat{g}_n(z)$、$h_n(z)$ 和 $\hat{h}_n(z)$ 是对从 $(0,0)$ 到 (n,n) 的,长度为 $2n$ 的所有格通路求和的 $\sum z^{\text{通路的总权数}}$,其中的权如图 11 所示定义,它受到通路上诸顶点的某些限制:对于 $h_n(z)$,没有限制,但对于 $g_n(z)$,这条通路必须回避所有使 $i>j$ 的顶点 (i,j);$\hat{h}_n(z)$ 和 $\hat{g}_n(z)$ 的定义与此类似,但要排除 $0<i<n$ 的所有顶点 (i,i)。于是

$$g_0(z)=1,\ g_1(z)=z,\ g_2(z)=z^3+z^2;\ \hat{g}_1(z)=z, \hat{g}_2(z)=z^3$$

$$h_0(z)=1,\ h_1(z)=z+1,\ h_2(z)=z^3+z^2+3z+1$$

$$\hat{h}_1(z)=z+1,\ \hat{h}_2(z)=z^3+z$$

试求出定义这些函数的递归关系,并使用递归关系来证明

$$h''_n(1)+h'_n(1)=\frac{7n^3+4n^2+4n}{30}\binom{2n}{n}$$

(因此容易找出 $\{1,2,\cdots,2n\}$ 的一个随机 2 有序排列中,反序个数方差的精确公式;它渐近于 $\left(\frac{7}{30}-\frac{\pi}{16}\right)n^3$)。

16. [*M24*] 求 $\{1,2,\cdots,n\}$ 的一个 h 有序排列中,反序的极大个数的公式。当增量满足整除性条件(5)时,在算法 D 中最多能有多少次移动?

17. [*M21*] 说明,若对 $t>s\geqslant 0$ 有 $N=2^t$ 和 $h_s=2^s$,则存在 $\{1,2,\cdots,N\}$ 的惟一排列,使由算法 D 执行的移动操作个数取极大值。试求描述这个排列的一个简单方式。

18. [*HM24*] 对于很大的 N,和数(6)可被估计为

$$\frac{1}{4}\frac{N^2}{h_{t-1}}+\frac{\sqrt{\pi}}{8}\left(\frac{N^{3/2}h_{t-1}^{1/2}}{h_{t-2}}+\cdots+\frac{N^{3/2}h_1^{1/2}}{h_0}\right)$$

当 N 和 t 固定且 $h_0=1$ 时，$h_{t-1},\cdots,h_0$ 的什么实数值使这个表达式取极小值？

▶19. [*M25*] 在程序 D 的计时分析中，当增量满足整除性条件(5)时，量 A 的平均值是多少？

20. [*M20*] 证明定理 K 可从引理 L 推得。

21. [*M25*] 设 h 和 k 是互素的正整数，并且说一个整数是可生成的，如果对于某些非整数 x 和 y，它等于 $xh+yk$ 的话。试证明，n 是可生成的当且仅当 $hk-h-k-n$ 不是可生成的(由于 0 是最小可生成的整数，因此最大的不可生成的整数必定是 $hk-h-k$。由此得出，在既是 h 有序又是 k 有序的任何文件中，每当 $j-i\geqslant(h-1)(k-1)$ 时，有 $K_i\leqslant K_j$)。

22. [*M30*] 证明所有 $\geqslant 2^s(2^s-1)$ 的整数均可表示成

$$a_0(2^s-1)+a_1(2^{s+1}-1)+a_2(2^{s+2}-1)+\cdots$$

的形式，其中，诸 a_j 是非负整数。但 $2^s(2^s-1)-1$ 则不能这样表示。而且，恰有 $2^{s-1}(2^s+s-3)$ 个正整数是不能以这样的形式来表示的。

当量 2^k-1 被 2^k+1 代替时，试求在这个表示之下类似的公式。

▶23. [*M22*] 证明如果 h_{s+2} 和 h_{s+1} 互素，则当算法 D 使用增量 h_s 时，移动次数是 $O(Nh_{s+2}h_{s+1}/h_s)$。提示，见习题 21。

24. [*M42*] 证明，在指数 3/2 已不能再降低的前提下，定理 P 是最好的。

▶25. [*M22*] $\{1,2,\cdots,n\}$ 有多少排列既是 3 有序的又是 2 有序的？在这样一个排列中反序的极大个数是多少？在所有这样的排列中反序的总数是什么？

26. [*M35*] 如果 N 个元素的一个文件是 3 有序、5 有序和 7 有序的，则它能有多于 N 个反序吗？当 N 很大时，试估计极大的反序个数。

27. [*M41*] (Bjorn Poonen) (a)试证明，如果算法 D 中的增量 h_s 中有 m 个小于 $N/2$，则存在一个常数 c，使得在最坏情况下的运行时间是 $\Omega(N^{1+c/\sqrt{m}})$。(b)对于所有增量序列，结果最坏的运行时间是 $\Omega(N(\log N/\log\log N)^2)$。

28. [*15*] 从程序 D 的观点看，以及考虑平均的总运行时间，表 6 中所示的增量序列哪个是最好的？

29. [*40*] 对于 $N=1000$ 以及各种 t 值，试求 $h_{t-1},\cdots,h_1,h_0$ 的经验值，对于它们，B_{ave} 是你所能达到的最小值。

30. [*M23*] (V. Pratt)　如果 Shell 排序中的增量集合是 $\{2^p3^q\mid 2^p3^q<N\}$，试证明扫描次数近似于 $\frac{1}{2}(\log_2 N)(\log_3 N)$，而每次扫描的移动次数至多是 $N/2$。事实上，如果对于任何一次扫描 $K_{j-h}>K_j$，则将总有 $K_{j-3h},K_{j-2h}\leqslant K_j<K_{j-h}\leqslant K_{j+h},K_{j+2h}$；所以可以简单地交换 K_{j-h} 和 K_j，并使 j 增加 $2h$，这节省了算法 D 的两次比较。提示：见习题 25。

▶31. [*25*] 为 Pratt 的排序算法写一个 MIX 程序(习题 30)。试借用类似于程序 D 中的那些量 A,B,S,T,N，来表达它的运行时间。

32. [*10*] 如果表 8 中的表插入排序一直进行到完成为止，则 $L_0,L_1,\cdots,L_{16}$ 的最后内容将是什么？

▶33. [*25*] 试求改进程序 L 的一种方法，使得主宰它的运行时间的是 $5B$ 而不是 $7B$，其中 B 是反序的数目。试讨论对于程序 S 的相应的改进。

34. [*M10*] 验证公式(14)。

35. [*21*] 遵循程序 M，写一个 MIX 程序，使得所有的表组合成一个表。你的程序应当与程序

L 中完全一样地来对 LINK 字段置值。

36. [*18*] 假设 MIX 的字节大小是 100,表 8 中 16 个键码的例子实际上是 503000,087000,512000,…,703000。当 $M=4$ 时试针对这组数据,确定出程序 L 和 M 的运行时间。

37. [*M25*] 设 $g_n(z)$ 是 n 个对象的一个随机排列中反序的生成函数(参考等式 5.1.1-(11))。设 $g_{NM}(z)$ 是程序 M 中量 B 的对应生成函数。证明

$$\sum_{N\geqslant 0} g_{NM}(z)\frac{M^N w^N}{N!} = \Big(\sum_{n\geqslant 0} g_n(z)\frac{w^n}{n!}\Big)^M$$

并使用这个公式导出 B 的方差。

38. [*HM23*] (R.M.Karp) 设 $F(x)$ 是一个概率分布的分布函数,且 $F(0)=0$ 及 $F(1)=1$。若键码 $K_1,K_2,\cdots,K_N$ 是从这个分布中随机地独立选择的,而且 $M=cN$,其中 c 是常数,$N\to\infty$,试证明当 F 充分光滑时,程序 M 的运行时间是 $O(N)$(当 $\lfloor MK\rfloor = j-1$ 时,一个键码 K 被插入到表 j 中;这以 $F(j/M)-F((j-1)/M)$ 的概率出现。正文中仅讨论了 $F(x)=x, 0\leqslant x\leqslant 1$ 的情况)。

39. [*HM16*] 如果一个程序的运行近似地需要 $A/M+B$ 个时间单位,并使用 $C+M$ 个内存单元,则怎样选择 M 可给出极小的空间×时间?

▶40. [*HM24*] 当 $N\to\infty$,对于固定的 α,$M=N/\alpha$ 时,试求在多个表插入中出现的自右至左极大值平均个数的渐近值,即等式(15)。借助于指数积分函数 $E_1(z)=\int_z^\infty e^{-t}dt/t$ 来表达你的答案,并且展开到 $O(N^{-1})$ 的绝对误差的范围内。

41. [*HM26*] (a)证明(10)的头 $\binom{k}{2}$ 个元素之和是 $O(\rho^{2k})$。(b)现在证明定理 I。

42. [*HM43*] 假定 $h\perp g$,当有 $t=3$ 个增量 h,g 和 1 时,试分析 Shell 排序的平均特性。头一遍扫描,即 h 排序,显然总共做了 $\frac{1}{4}N^2/h+O(N)$ 次移动。

a) 证明,第二次扫描,即 g 排序,做了 $\frac{\sqrt{\pi}}{8}(\sqrt{h}-1/\sqrt{h})N/g+O(hN)$ 次移动。

b) 证明,第三次扫描,即 1 排序,做了 $\psi(h,g)N+O(g^3h^2)$ 次移动,其中

$$\psi(h,g)=\frac{1}{2}\sum_{d=1}^{g-1}\sum_j\binom{h-1}{j}\Big(\frac{d}{g}\Big)^j\Big(1-\frac{d}{g}\Big)^{h-1-j}\Big|j-\Big\lfloor\frac{hd}{g}\Big\rfloor\Big|$$

▶43. [*25*] 通过使测试"$i>0$"在步骤 S4 中成为不必要的,习题 33 使用一个标记来加速算法 S。这个技巧对于算法 D 不适用。尽管如此,试证明在步骤 D5 中有一个容易的方法来避免对"$i>0$"的测试,由此加速 Shell 排序的内循环。

44. [*M25*] 如果 $\pi=a_1\cdots a_n$ 和 $\pi'=a_1'\cdots a_n'$ 是 $\{1,\cdots,n\}$ 的排序,对于 $1\leqslant i\leqslant j\leqslant n$,如果 $\{a_1,\cdots,a_j\}$ 的第 i 个最大的元素小于或等于 $\{a_1',\cdots,a_j'\}$ 的第 i 个最大的元素,则说 $\pi\leqslant\pi'$(换句话说,对于所有的 j,在头 j 个元素已被插入之后,如果 π 的直接插入排序按分量小于或等于 π' 的直接插入排序,则 $\pi\leqslant\pi'$)。

a) 如果在习题 5.1.1-12 的意义下,π 在 π' 之上,是否有 $\pi\leqslant\pi'$?

b) 如果 $\pi\leqslant\pi'$,是否有 $\pi^R\geqslant\pi'^R$?

c) 如果 $\pi\leqslant\pi'$,是否有 π 在 π' 之上?

5.2.2 通过交换进行排序

现在回过来讨论 5.2 节开头提到的第二类排序算法:"交换"或"换位"方法,它

系统地交换反序的元素对偶，直到不再存在这样的对偶为止。

直接插入算法 5.2.1S 的过程，可以看成是一个交换方法：我们基本上把每个新记录 R_j 同它左边的相邻者进行交换，直到它插入到适当的位置为止。所以，把排序方法分成各种类型，诸如“插入”、“交换”、“选择”等，其界线并不总是很分明的。在这一小节将讨论以交换为主要特征的 4 种排序方法，它们是：交换选择（“冒泡排序”）；合并交换（Batcher）的平行排序）；分划交换（Hoare）的“快速排序”）；以及基数交换。

冒泡排序　也许通过交换进行排序的最明显方法是比较 K_1 和 K_2，如果这些键码不是顺序的，就交换 R_1 和 R_2；然后对记录 R_2 和 R_3，R_3 和 R_4 等进行相同的工作。在这一系列操作期间，具有较大键码的记录势必向右移动，而且事实上具有最大键码的记录将移动到成为 R_N 为止。重复这个过程，就得到在位置 R_{N-1}、R_{N-2}等处的适当记录，使得所有记录最终排好序。

图 14 示出了对于 16 个键码 503 087 512…703 进行这种排序的方法。垂直地而不是水平地表示数的文件，并以 R_N 在顶部和 R_1 在底部，是适宜的。这个方法称为“冒泡排序”，因为较大的元素“上浮”到它们适当的位置，同“下沉排序”（即直接插入）相反。在下沉排序中，元素下沉到一个适当的位置。冒泡排序还有其它比较乏

	扫描 1	扫描 2	扫描 3	扫描 4	扫描 5	扫描 6	扫描 7	扫描 8	扫描 9
703	908	908	908	908	908	908	908	908	908
765	703	897	897	897	897	897	897	897	897
677	765	703	765	765	765	765	765	765	765
612	677	765	703	703	703	703	703	703	703
509	612	677	677	677	677	677	677	677	677
154	509	612	653	653	653	653	653	653	653
426	154	509	612	612	612	612	612	612	612
653	426	154	509	512	512	512	512	512	512
275	653	426	154	509	509	509	509	509	509
897	275	653	426	154	503	503	503	503	503
170	897	275	512	426	154	426	426	426	426
908	170	512	275	503	426	154	275	275	275
061	512	170	503	275	275	275	154	170	170
512	061	503	170	170	170	170	170	154	154
087	503	061	087	087	087	087	087	087	087
503	087	087	061	061	061	061	061	061	061

图 14　进行中的冒泡排序

味的名字,例如称为“交换选择”或“扩散”等。

在每次扫过这文件之后,不难看出,上边的所有记录,包括最后被交换的那个记录在内,都必然处于它们的最后位置上,所以在随后的扫描中无须考察它们。图 14 中的水平线根据这个观点示出了这个排序的过程;例如注意,在第 4 趟之后,已知又有 5 个元素处于最后位置。在最后一趟扫描时已全然不实施任何交换了。通过这些考察,我们就容易地系统阐述这个算法了。

算法 B(冒泡排序) 适当地重新排列记录 $R_1,\cdots,R_N$;在排序完成之后,它们的键码将是有序的,即 $K_1\leqslant\cdots\leqslant K_N$。

B1.[初始化 BOUND] 置 BOUND$\leftarrow N$(BOUND 是尚不知其记录是否已处于最终位置上的最高下标;这表示此刻我们尚一无所知)。

B2.[对 j 进行循环] 置 $t\leftarrow 0$,对 $j=1,2,\cdots$,BOUND -1 执行步骤 B3,然后转到步骤 B4(如果 BOUND$=1$,则意味着直接转到 B4)。

B3.[比较/交换 $R_j:R_{j+1}$] 如果 $K_j>K_{j+1}$,则交换 $R_j\longleftrightarrow R_{j+1}$,并置 $t\leftarrow j$。

B4.[是否还要交换?] 如果 $t=0$,则此算法终止。否则置 BOUND$\leftarrow t$,并返回到步骤 B2。 ▌

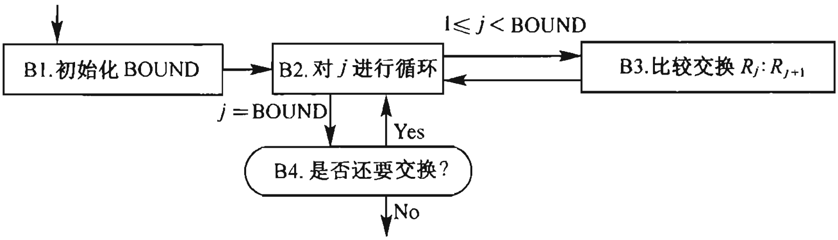


图 15 冒泡排序的框图

程序 B(冒泡排序) 如同本章前面的 MIX 程序一样,我们假定有待排序的项目是在单元 INPUT + 1 到 INPUT + N 中。rI1≡t;rI2≡j。

```
START   ENT1   N             1   B1.初始化 BOUND 。t←N
1H      ST1    BOUND(1:2)    A   BOUND←t
        ENT2   1             A   B2.对 j 进行循环。j←1
        ENT1   0             A   t←0
        JMP    BOUND         A   如果 j≥BOUND,则退出
3H      LDA    INPUT,2       C   B3.比较/交换 R_j:R_{j+1}
        CMPA   INPUT+1,2     C
        JLE    2F            C   如果 K_j≤K_{j+1},则不交换
        LDX    INPUT+1,2     B   R_{j+1}
        STX    INPUT,2       B     →R_j
```

```
        STA    INPUT+1,2    B      (旧的 R_j)→R_{j+1}
        ENT1   0,2          B      t←j
2H      INC2   1            C      j←j+1
BOUND   ENTX   -*,2         A+C    rX←j-BOUND(被修改的指令)
        JXN    3B           A+C    对 1≤j<BOUND 执行步骤 B3
4H      J1P    1B           A      B4.是否还要交换？如果 t>0,则转 B2 ▌
```

冒泡排序的分析　分析一下算法 B 的运行时间是很有益处的。在计时中涉及了 3 个量:扫描次数 A;交换次数 B;以及比较次数 C。如果输入的键码是不同的,并处于随机次序,则可以假定它们形成$\{1,2,\cdots,n\}$的一个随机排列。从反序表(5.1.1 小节)的思想可推出一个容易的方法,来描述在一个冒泡排序中的每次扫描的效果。

定理 I　*设 $a_1,a_2,\cdots,a_n$ 是$\{1,2,\cdots,n\}$的一个排列,并设 $b_1,b_2,\cdots,b_n$ 是对应的反序表。如果冒泡排序算法B的一次扫描把 $a_1a_2\cdots a_n$ 变成排列$a'_1a'_2\cdots a'_n$,则从 $b_1b_2\cdots b_n$ 中把每个非零项减1,就得到对应的反序表 $b'_1b'_2\cdots b'_n$。*

证明　如果有一个较大的元素居于 a_i 之前,则居于前面的最大元素就同它进行交换,所以 b_{a_j} 减 1。另一方面,如果没有较大的元素居于 a_i 之前,它就决不同一个较大的元素交换,所以 b_{a_i} 保持为零。　▌

于是,通过研究在诸次扫描中的反序表序列,即可看出在进行冒泡排序期间发生的情况。例如,对应于图 14 的逐次的反序表是:

$$
\begin{array}{ll}
 & 3\ 1\ 8\ 3\ 4\ 5\ 0\ 4\ 0\ 3\ 2\ 2\ 3\ 2\ 1\ 0 \\
\text{扫描 1} & \\
 & 2\ 0\ 7\ 2\ 3\ 4\ 0\ 3\ 0\ 2\ 1\ 1\ 2\ 1\ 0\ 0 \\
\text{扫描 2} & \\
 & 1\ 0\ 6\ 1\ 2\ 3\ 0\ 2\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 0 \\
\text{扫描 3} & \\
 & 0\ 0\ 5\ 0\ 1\ 2\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0
\end{array}
\tag{1}
$$

等等。因此如果 $b_1b_2\cdots b_n$ 是输入排列的反序表,我们必然有

$$A = 1 + \max(b_1,b_2,\cdots,b_n) \tag{2}$$

$$B = b_1 + b_2 + \cdots + b_n \tag{3}$$

$$C = c_1 + c_2 + \cdots + c_A \tag{4}$$

其中 c_j 是在第 j 次扫描开始时 BOUND－1 的值。借助于反序表来表示,有

$$c_j = \max\{b_i + i \mid b_i \geqslant j-1\} - j \tag{5}$$

(见习题 5)。在例(1)中,因此有 $A=9,B=41,C=15+14+13+12+7+5+4+3+2=75$。对于图 14,总的 MIX 排序时间是 960 u。

B(在一个随机排列中的反序的总数)的分布我们已经非常清楚的,所以只剩下 A 和 C 有待分析。

$A \leqslant k$ 的概率是 $1/n!$ 乘上没有 $\geqslant k$ 的分量的反序表数目,当 $1 \leqslant k \leqslant n$ 时即是 $k^{n-k}k!$。因此恰好要求 k 次扫描的概率是

$$A_k = \frac{1}{n!}(k^{n-k}k! - (k-1)^{n-k+1}(k-1)!) \tag{6}$$

由此可计算均值 $\sum kA_k$;作部分求和,它就是

$$A_{\text{ave}} = n + 1 - \sum_{h=0}^{z} \frac{k^{n-k}k!}{n!} = n + 1 - P(n) \tag{7}$$

其中 $P(n)$ 是在等式 1.2.11.3-(24)中求出的其渐近值为 $\sqrt{\pi n/2} - \frac{2}{3} + O(1/\sqrt{n})$ 的函数。E.H.Friend 在 *JACM* **3** (1956),150 中未加证明地指出了公式(7);它的证明由 Howard B.Demuth 给出[博士论文(Stanford University,1956 年 10 月),64~68]。关于 A 的标准差,见习题 7。

比较的总数 C 要更难处理些,我们将仅仅考虑 C_{ave}。对于固定的 n,令 $f_j(k)$ 是对于 $1 \leqslant i \leqslant n$,使得 $b_i < j-1$ 或 $b_i + i - j \leqslant k$ 的反序表 $b_1 \cdots b_n$ 的数目;于是

$$f_j(k) = (j+k)!(j-1)^{n-j-k}, \quad 0 \leqslant k \leqslant n-j \tag{8}$$

(见习题 8)。(5)中 c_j 的平均值是$(\sum k(f_j(k) - f_j(k-1)))/n!$;进行部分求和,然后对 j 求和,即导出公式

$$C_{\text{ave}} = \binom{n+1}{2} - \frac{1}{n!}\sum_{\substack{1 \leqslant j \leqslant n \\ 0 \leqslant k \leqslant n-j}} f_j(k) = \binom{n+1}{2} - \frac{1}{n!}\sum_{0 \leqslant r < s \leqslant n} s!\, r^{n-s} \tag{9}$$

这里渐近值不容易确定。因而将在本节末再回过头来研究它。

为了总结对于冒泡排序的分析,以上和以下所导出的公式可以写成如下的结果:

$$A = (\min 1, \text{ave } N - \sqrt{\pi N/2} + O(1), \max N) \tag{10}$$

$$B = (\min 0, \text{ave } \frac{1}{4}(N^2 - N), \max \frac{1}{2}(N^2 - N)) \tag{11}$$

$$C = (\min N - 1, \text{ave } \frac{1}{2}(N^2 - N\ln N - (\gamma + \ln 2 - 1)N) + O(\sqrt{N}),$$
$$\max \frac{1}{2}(N^2 - N)) \tag{12}$$

在每种情况下,当输入已处于有序时即出现极小值;所以 MIX 运行时间是 $8A + 7B + 8C + 1 = (\min 8N + 1, \text{ave } 5.75N^2 + O(N\log N), \max 7.5N^2 + 0.5N + 1)$。

冒泡排序的改进 为分析冒泡排序做了大量的工作;尽管用于这些计算的技术是有教益的,但结果却是令人失望的,因为它告诉我们,冒泡排序全然不是非常好的。同直接插入相比(算法 5.2.1S),冒泡排序需要相当复杂的程序,而且花费超过两倍的时间!

冒泡排序很容易就可找出某些缺陷。例如，在图 14 中，扫描 4 中的头一个比较是多余的，扫描 5 中的头两个，以及扫描 6 和扫描 7 中的头三个也是多余的。还要注意，每次扫描时元素向左移动决不能多于一步；所以，如果开始时最小的项目正好靠近右端，则我们就被迫进行最大次数的比较。这就提出了“鸡尾混合排序”，它以相反的方向交替地进行扫描（见图 16）。这个方法略微减少了进行比较的平均次数。在这方面 K.E. Iverson [*A Programming Language* (Wiley, 1962), 218～219] 已经进行了有趣的考察：对于相反方向的两次连续扫描，如果 jR_j 和 R_{j+1} 彼此不交换的一个下标，则 R_j 和 R_{j+1} 必须都处于它们最后的位置，而且它们都不必进行任何随后的比较。例如，从左到右遍历 4 3 2 1 8 6 9 7 5 得出 3 2 1 4 6 8 7 5 9；在 R_4 和 R_5 之间没有出现交换，从右到左遍历后一个排列，R_4 仍然小于（新的）R_5，所以可以立即得出结论，R_4 和 R_5 无须进行任何进一步的比较。

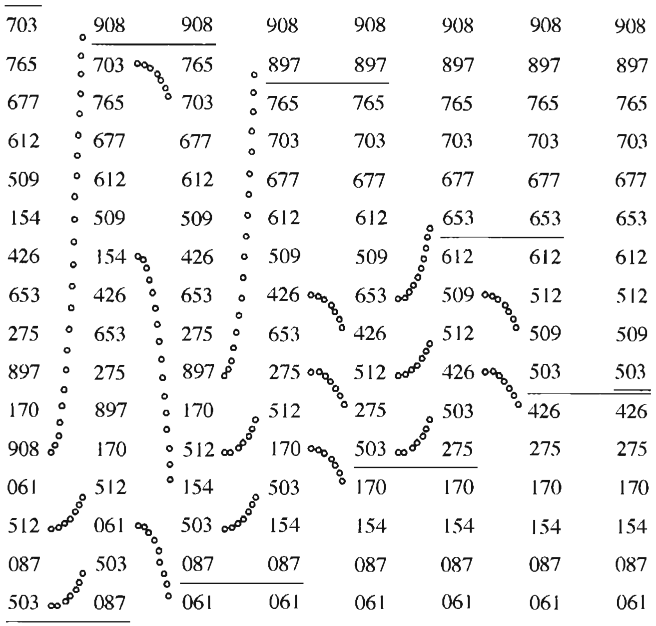


图 16 “鸡尾混合排序”

但是这些改进中没有一个能导出比直接插入更好的算法；而且我们已经知道，对于很大的 N，直接插入是不适当的。另一个想法是消除大多数的交换；因为在一个交换期间大多数元素都只不过左移一步，故通过移动下标的起点，换一种方式考察这个数组，就可以得到同样的效果！但得到的算法仍不比后面将要研究的直接选择算法 5.2.3S 更好。

简言之，冒泡排序除开它迷人的名字和导致了某些有趣的理论问题这一事实之外，似乎没有什么值得推荐的。

Batcher 平行法 如果想要有一个其运行时间比 N^2 的阶数快的交换算法，就需要选择某个不相邻的键码对偶(K_i, K_j)进行比较；否则，就需要有像原来排列的所有反序那样多的交换，而反序的平均数为$\frac{1}{4}(N^2-N)$。考察可能的交换，对比较序列进行编程的一个巧妙方法是 1964 年由 K.E.Batcher 发现的[*Proc. AFIPS Spring Joint Computer Conference* **32** (1968),307～314]。他的方法并不是十分显然的，由于只进行相当小的比较，因此需要相当错综的论证，才能证明它是正确的。我们将讨论两个证明，一个在这一节中，另一个在 5.3.4 小节中。

Batcher 的排序方案同 Shell 的排序有些相似，但是比较是在一种新奇的途径下完成的，以使交换的扩散成为不必要。例如，可以把表 1 与表 5.2.1-3 进行类比；Batcher 方法达到了 8,4,2 和 1 排序的效果，但是这些比较都不重叠。因为 Batcher 的算法实质上是合并排好序的子序列对偶，故可以称做"合并交换排序"。

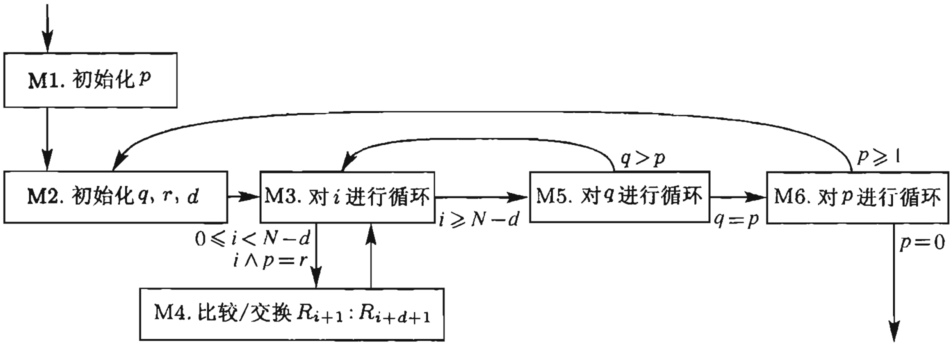


图 17 算法 M

算法 M(合并交换) 适当地重新排列记录 $R_1, \cdots, R_N$；在完成排序之后，它们的键码将是有序的：$K_1 \leqslant K_2 \leqslant \cdots \leqslant K_N$。假定 $N \geqslant 2$。

M1.[初始化 p] 置 $p \leftarrow 2^{t-1}$，其中 $t=\lceil \lg N \rceil$是使得 $2^t \geqslant N$ 的最小整数(将对 $p=2^{t-1}, 2^{t-2}, \cdots, 1$ 实施步骤 M2 到 M5)。

M2.[初始化 q, r, d] 置 $q \leftarrow 2^{t-1}, r \leftarrow 0, d \leftarrow p$。

M3.[对 i 进行循环] 对于使得 $0 \leqslant i < N-d$ 和 $i \wedge p = r$ 的所有 i，执行步骤 M4。然后转到步骤 M5(这里 $i \wedge p$ 指的是 i 和 p 二进表示的"逻辑与"；对每个二进位来说，除非 i 和 p 两数在该位上的值都为 1，否则结果为零。例如 $13 \wedge 21 = (1101)_2 \wedge (10101)_2 = (00101)_2 = 5$。这时，$d$ 是 p 的奇数倍，而且 p 是 2 的一个乘方，使得 $i \wedge p \neq (i+d) \wedge p$；由此得出，步骤 M4 的动作可以以任意次序，甚至同时对所有有关的 i 来完成)。

M4.[比较/交换 $R_{i+1}:R_{i+d+1}$] 如果 $K_{i+1} > K_{i+d+1}$，则交换 $R_{i+1} \leftrightarrow R_{i+d+1}$。

M5.[对 q 进行循环] 如果 $q \neq p$，则置 $d \leftarrow q-p, q \leftarrow q/2, r \leftarrow p$，并返回步骤 M3。

M6.［对 p 进行循环］（这时，排列 $K_1K_2\cdots K_N$ 是 p 有序的）置 $p\leftarrow\lceil p/2\rceil$。如果 $p>0$，则返回到 M2。 ▌

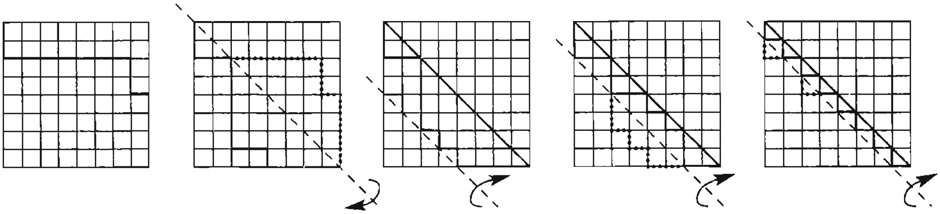
表 1 对于 $N=16$ 说明了本方法。注意，这个算法实质上先对 $R_1,R_3,R_5,\cdots$ 和 $R_2,R_4,R_6\cdots$ 独立地进行排序；然后，对 $p=1$ 实施步骤 M2 到 M5，为的是把两个排好序的序列合并在一起，从而完成对 N 个元素的排序。

表 1　合并交换排序（Batcher 方法）

p	q	r	d																
				503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703
8	8	0	8																
				503	087	154	061	612	170	765	275	653	426	512	509	908	677	897	703
4	8	0	4																
				503	087	154	061	612	170	765	275	653	426	512	509	908	677	897	703
4	4	4	4																
				503	087	154	061	612	170	512	275	653	426	765	509	908	677	897	703
2	8	0	2																
				154	061	503	087	512	170	612	275	653	426	765	509	897	677	908	703
2	4	2	6																
				154	061	503	087	512	170	612	275	653	426	765	509	897	677	908	703
2	2	2	2																
				154	061	503	087	512	170	612	275	653	426	765	509	897	677	908	703
1	8	0	1																
				061	154	087	503	170	512	275	612	426	653	509	765	677	897	703	908
1	4	1	7																
				061	154	087	503	170	512	275	612	426	653	509	765	677	897	703	908
1	2	1	3																
				061	154	087	275	170	426	503	509	512	653	612	703	677	897	765	908
1	1	1	1																
				061	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908

为了证明在算法 M 中描述的比较/交换的奇妙序列，实际上能对所有可能的输入文件 $R_1R_2\cdots R_N$ 排序，仅仅需要证明，当 $p=1$ 时步骤 M2 到 M5 将合并所有的 2 有序文件 $R_1R_2\cdots R_N$。为此，可以使用 5.2.1 小节的格盘通路方法（参见图 11）；$\{1,2,\cdots,N\}$ 的每个 2 有序排列惟一地对应于在一个格盘图示中从 $(0,0)$ 到 $(\lceil N/2\rceil,\lfloor N/2\rfloor)$ 的一条通路。图 18(a) 示出了 $N=16$ 的一个例子，对应于排列 1 3 2 4 10 5 11 6 13 7 14 8 15 9 16 12。当对于 $p=1,q=2^{t-1},r=0,d=1$ 执行步骤 M3 时，其效果是比较（可能还有交换）$R_1:R_2,R_3:R_4$，等等。这个操作对应于格盘通路的一

个简单变换,即在必要时相对于对角线"折叠"该通路,以使它决不落在对角线的上面部分(见图 18(b)和习题 10 中的证明)。步骤 M3 以后的迭代有 $p=r=1$,以及 $d=2^{t-1}-1,2^{t-2}-1,\cdots,1$;它们的效果是比较/交换 $R_2:R_{2+d}$,$R_4:R_{4+d}$,等等。而且也有一个简单的格盘解释:把这条通路相对于对角线下方$\frac{1}{2}(d+1)$个单位处的一条线作"折叠",见图 18(c)和(d);最后得到了图 18(e)中的通路,它对应于一个完全排好序的排列。这就完成了关于 Batcher 算法是正确的一个"几何证明";我们可以称它为通过折叠进行排序!



(a) (b) (c) (d) (e)

图 18 Batcher 方法的一个几何解释,$N=16$

算法 M 的一个 MIX 程序出现于习题 12 中。可惜,为控制比较序列所需要的簿记工作的量是相当大的,所以这个程序比我们所看到的其它方法效率都更低。但有一个重要的特性可补救它的不足:在允许进行并行计算的计算机或逻辑网络上,由步骤 M3 的迭代所确定的所有比较/交换都可以同时地完成。通过这样的并行操作,排序在$\frac{1}{2}\lceil \lg N\rceil(\lceil \lg N\rceil+1)$步内完成,因此这大体同任何已知的一般方法一样快。例如,*对于 1024 个元素使用 Batcher 方法可在仅仅 55 个平行步骤内进行排序*。与其最接近的竞争者是 Pratt 方法(见习题 5.2.1-30),它使用 40 或 73 步,这看我们如何计数;如果我们允许重叠的比较,只要不允许重叠的交换,则 Pratt 方法只需要 40 个比较/交换循环就可对 1024 个元素进行排序。进一步的说明,见 5.3.4 小节。

快速排序 Batcher 方法中的比较序列是预先确定的;每次比较相同的键码对偶,而不管关于这个文件从以前的比较中已经知道些什么。在冒泡排序中,尽管算法 B 有限地利用了以前的知识来减少它在这个文件右端的工作,但大体上仍是如此。现在我们转到一种十分不同的策略,它使用每个比较的结果来确定下一次要比较什么键码。这样一种策略对于并行计算是不合适的,但在顺序工作的计算机上却是十分有效的。

下列方法的基本思想是取一个记录,比如说 R_1,并把它移动到在排了序的文件中它将占据的最终位置,比如说位置 s。在确定这最终位置的同时,也重新排列其它的记录,使得带有较大键码的记录都位于 s 的右边,带有较小键码的则位于 s 的左边。这样一来文件就将以这样一个方式被划分,即原来的排序问题归结成两个较简单的问题,就是对 $R_1\cdots R_{s-1}$ 排序,以及(独立地)对 $R_{s+1}\cdots R_N$ 排序。可以应用同一技术到这些子文件中的每一个上,直到这一作业完成时为止。

把文件分划成左和右子文件有若干个方法。下面由 R. Sedgewick 给出的方案，似乎是最好的，其原因当分析此算法时就会清楚了。保持两个指针 i 和 j，而且开始时 $i=2$ 和 $j=N$。如果 R_i 在分划之后最终要成为左子文件的一部分(通过比较 K_i 和 K_1 我们可知道这一点)，则 i 增 1，并且继续进行直到遇到一个属于右子文件的记录 R_i 为止。类似地，j 减 1 直到遇到属于左子文件的一个记录 R_j 为止。如果 $i<j$，则交换 R_i 同 R_j；然后以同样方式处理下一记录，“两头点蜡”直到 $i \geq j$。通过交换 R_j 同 R_1，最终地完成这个分划。例如，试看对于 16 个数的文件所发生的情况(为了指出 i 和 j 的位置，K_i 和 K_j 已经以黑体字标出)。

		i ↓														j ↓
初始文件	[503	**087**	512	061	908	170	897	275	653	426	154	509	612	677	765	**703**]
第 1 次交换	503	087	**512**	061	908	170	897	275	653	426	**154**	509	612	677	765	703
第 2 次交换	503	087	154	061	**908**	170	897	275	653	**426**	512	509	612	677	765	703
第 3 次交换	503	087	154	061	426	170	**897**	**275**	653	908	512	509	612	677	765	703
指针交叉	503	087	154	061	426	170	**275**	**897**	653	908	512	509	612	677	765	703
分划后的文件	[275	087	154	061	426	170]	**503**	[**897**	653	908	512	509	612	677	765	703]
							↑ j	↑ i								

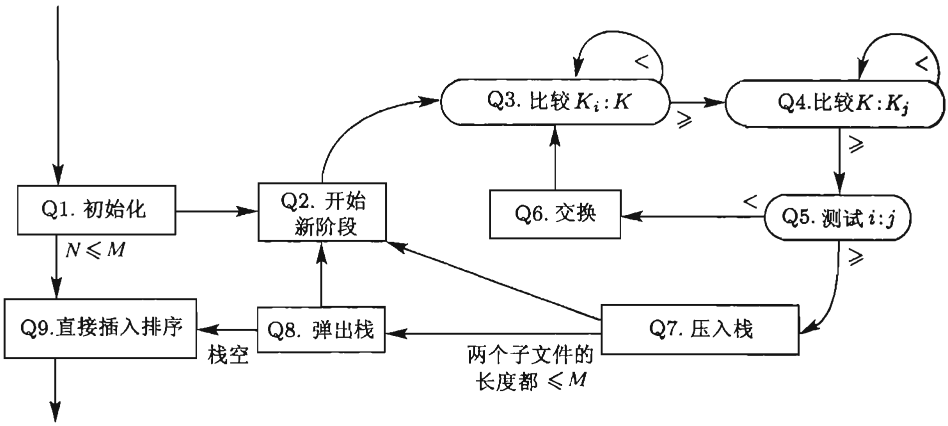


图 19　分划交换排序(快速排序)

表 2 示出了怎样通过这种方法，用 11 个阶段把我们的示例文件完全排好序。方括弧指出了仍然需要加以排序的子文件；粗方括弧标出当前感兴趣的子文件。在一台计算机内，当前的子文件可以通过边界值(l,r)表示，而其它的子文件通过附加的对偶(l_k,r_k)的一个栈来表示。每次分划这个文件，就把较长的子文件压入栈，并对另一个较短的子文件着手进行工作，直到得到非常容易排序的短文件为止；这个策略确保栈决不包含多于 lg N 个项目(见习题 20)。

表 2 快速排序

																(l,r)	栈
〚503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703〛	(1,16)	–
〚275	087	154	061	426	170〛	503	[897	653	908	512	509	612	677	765	703]	(1,6)	(8,16)
〚170	087	154	061〛	275	426	503	[897	653	908	512	509	612	677	765	703]	(1,4)	(8,16)
〚061	087	154〛	170	275	426	503	[897	653	908	512	509	612	677	765	703]	(1,3)	(8,16)
061	〚087	154〛	170	275	426	503	[897	653	908	512	509	612	677	765	703]	(2,3)	(8,16)
061	087	154	170	275	426	503	〚897	653	908	512	509	612	677	765	703〛	(8,16)	–
061	087	154	170	275	426	503	〚765	653	703	512	509	612	677〛	897	908	(8,14)	–
061	087	154	170	275	426	503	〚677	653	703	512	509	612〛	765	897	908	(8,13)	–
061	087	154	170	275	426	503	〚509	653	612	512〛	677	703	765	897	908	(8,11)	–
061	087	154	170	275	426	503	509	〚653	612	512〛	677	703	765	897	908	(9,11)	–
061	087	154	170	275	426	503	509	〚512	612〛	653	677	703	765	897	908	(9,10)	–
061	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908	–	–

刚才描述的排序过程可以称做分划交换排序，它是由 C. A. R. Hoare 提出来的。在所有已经发表过的排序方法中，他的有趣的论文[*Comp.J.* **5** (1962)，10～15]中所讨论的方法是研究最深的之一。Hoare 把他的方法称为"快速排序"；这样一个名称并非不当，因为，这个计算的内循环在大多数计算机上都是极快的。在一个给定的阶段，所有比较都针对同一个键码进行，所以这个键码可保存在一个寄存器中。在比较之间只须改变一个下标。而且数据移动的数量是十分合理的；例如，表 2 中的计算仅做 17 次交换。

额外的簿记(用来控制 i、j 和栈)并不困难，但它使得快速排序分划过程最适合于很大的 N；因此在子文件已经变短之后，下列算法使用另外的策略。

算法 Q(快速排序) 适当地重新安排记录 $R_1,\cdots,R_N$；在排序完成之后，它们的键码将是有序的，即 $K_1\leqslant\cdots\leqslant K_N$。需要有至多$\lfloor\lg N\rfloor$个项目的一个辅助栈作为临时存储。本算法遵循上面正文中所述的"快速排序"分划过程，但为了提高效率已稍做修改：

a) 假定存在人为的键码 $K_0=-\infty$ 和 $K_{N+1}=+\infty$ 使得

$$K_0\leqslant K_i\leqslant K_{N+1}\quad 对于\ 1\leqslant i\leqslant N \tag{13}$$

(允许相等)。

b) 具有 M 个或更少个元素的子文件，直到这个过程真正结束之前都保持未排序。然后使用直接插入的一次扫描产生最后的次序。其中 $M\geqslant 1$ 是一个参数，它应

像以下正文所述的那样进行选择(由 R. Sedgewick 给出的这一思想,可节省某些开销,如果我们直接应用直接插入到每个小的子文件,则这个开销将是需要的,除非访问的局部性是重要的)。

c) 交换具有相等键码的记录,尽管这样做并非严格必要(由 R.C. Singeton 给出的这一思想,保持了内循环的快速,而且当存在相等的元素时,帮助分开近乎一半的子文件;参见习题 18)。

Q1.[初始化]　如果 $N \leqslant M$,转到步骤 Q9。否则置栈为空,并置 $l \leftarrow 1, r \leftarrow N$。

Q2.[开始新阶段]　(现在希望对子文件 $R_l \cdots R_r$ 进行排序;从算法的本性出发,有 $r \geqslant l+M$,而且对于 $l \leqslant i \leqslant r, K_{l-1} \leqslant K_i \leqslant K_{r+1}$) 置 $i \leftarrow l, j \leftarrow r+1$;并置 $K \leftarrow K_l$(下文讨论了对 K 另外的选择,它们可能是更好的)。

Q3.[比较 $K_i : K$]　(此时这个文件已被重排,使得

$$\text{对于 } l-1 \leqslant k \leqslant i,\quad K_k \leqslant K,\qquad \text{对于 } j \leqslant k \leqslant r+1,\quad K \leqslant K_k \tag{14}$$

且 $l \leqslant i < j$) i 增 1;其后如果 $K_i < K$,则重复该步骤(由于 $K_j \geqslant K$,此迭代必定以 $i \leqslant j$ 结束)。

Q4.[比较 $K : K_j$]　j 减 1;其后如果 $K < K_j$,则重复此步骤(由于 $K \geqslant K_{i-1}$,该迭代必定以 $j \geqslant i-1$ 结束)。

Q5.[测试 $i : j$]　(这时,除对于 $k=i$ 和 $k=j$ 外,(14)成立;而且 $K_i \geqslant K \geqslant K_j$,$r \geqslant j \geqslant i-1 \geqslant l$) 如果 $j \leqslant i$,交换 $R_l \leftrightarrow R_j$,并转到步骤 Q7。

Q6.[交换]　交换 $R_i \leftrightarrow R_j$ 并转回步骤 Q3。

Q7.[压入栈]　(现在子文件 $R_l \cdots R_i \cdots R_r$ 已经被分划,使得对于 $l-1 \leqslant k \leqslant j$,$K_k \leqslant K_j$ 且对于 $j \leqslant k \leqslant r+1$,$K_j \leqslant K_k$)如果 $r-j \geqslant j-l > M$,则把$(j+1, r)$插到栈顶上,置 $r \leftarrow j-1$,并转到 Q2。如果 $j-l > r-j > M$,则把$(l, j-1)$插入到栈顶上,置 $l \leftarrow j+1$,并转到 Q2(在栈上的每个项目(a,b),是在某个将来的时刻对子文件 $R_a \cdots R_b$ 排序的一个请求)。否则如果 $r-j > M \geqslant j-l$,置 $l \leftarrow j+1$ 并转到 Q2;或者如果 $j-l > M \geqslant r-j$,置 $r \leftarrow j-1$ 并转到 Q2。

Q8.[弹出栈]　如果这个栈不空,则撤销它顶部的项(l', r'),置 $l \leftarrow l', r \leftarrow r'$,并返回步骤 Q2。

Q9.[直接插入排序]　对于 $j = 2, 3, \cdots N$,如果 $K_{j-1} > K_j$,则进行下列操作:置 $K \leftarrow K_j, R \leftarrow R_j, i \leftarrow j-1$;然后 $R_{i+1} \leftarrow R_i$ 并且 $i \leftarrow i-1$,一次或多次直到 $K_i \leqslant K$;然后置 $R_{i+1} \leftarrow R$(这是如同习题 5.2.1-10 和答案 5.2.1-33 所提议的那样修改过了的算法 5.2.1S。如果 $M=1$,则步骤 Q9 可以省略。警

告：最后的直接插入可以掩盖步骤 Q1～Q8 中的失误；不要仅仅由于一个实现给出正确的答案就信任它）。▍

对应的 MIX 程序是相当长的，但并不复杂。事实上，大部分代码是专门用于步骤 Q7 的，它仅仅以一种非常直截了当的方式来摆弄那些变量。

程序 Q（快速排序）　有待排序的记录在单元 INPUT + 1 到 INPUT + N 中；假定单元 INPUT 和 INPUT + N + 1 分别包含 MIX 中最小和最大可能的值。栈保存在单元 STACK + 1，STACK + 2，…中；关于栈所需单元的精确数目，见习题 20。rI2≡l，rI3≡r，rI4≡i，rI5≡j，rI6≡栈的大小，rA≡K≡R。我们假定 $N>M$。

```
A       EQU    2:3                   栈元素的第一个分量
B       EQU    4:5                   栈元素的第二个分量
START   ENT6   0            1        Q1. 初始化。置栈为空
        ENT2   1            1        置 l←1
        ENT3   N            1        置 r←N
2H      ENT5   1,3          A        Q2. 开始新阶段。置j←r+1
        LDA    INPUT,2      A        置 K←K_l
        ENT4   1,2          A        置 i←l+1
        JMP    0F           A        转 Q3 并省略"i←i+1"
6H      LDX    INPUT,4      B        Q6. 交换
        ENT1   INPUT,4      B
        MOVE   INPUT,5      B
        STX    INPUT,5      B        R_i←→R_j
3H      INC4   1            C'-A     Q3. 比较 K_i:K。置 i←i+1
0H      CMPA   INPUT,4      C'
        JG     3B           C'       如果 K>K_i，则重复
4H      DEC5   1            C-C'     Q4. 比较 K:K_j。置 j←j-1
        CMPA   INPUT,5      C-C'
        JL     4B           C-C'     如果 K<K_j，则重复
5H      ENTX   0,5          B+A      Q5. 测试 i:j
        DECX   0,4          B+A
        JXP    6B           B+A      如果 j>i，则转 Q6
        LDX    INPUT,5      A
        STX    INPUT,2      A        R_l←R_j
        STA    INPUT,5      A        R_j←R
```

```
7H   ENT4   0,3             A            Q7.压入栈
     DEC4   M,5             A            rI4←r-j-M
     ENT1   0,5             A
     DEC1   M,2             A            rI1←j-l-M
     ENTA   0,4             A
     DECA   0,1             A
     JANN   1F              A            如果 r-j≥j-l,则跳转
     J1NP   8F              A'           如果 M≥j-l>r-j,则转 Q8
     J4NP   3F              S'+A''       如果 j-l>M≥r-j,则跳转
     INC6   1               S'           (现在 j-l>r-j>M)
     ST2    STACK,6(A)      S'
     ENTA   -1,5            S'
     STA    STACK,6(B)      S'           (l,j-1)⇒栈
4H   ENT2   1,5             S'+A'''      置 l←j+1
     JMP    2B              S'+A'''      转 Q2
1H   J4NP   8F              A-A'         如果 M≥r-j≥j-l,则转 Q8
     J1NP   4B              S-S'+A'''    如果 r-j>M≥j-l,则跳转
     INC6   1               S-S'         (现在 r-j≥j-l>M)
     ST3    STACK,6(B)      S-S'
     ENTA   1,5             S-S'
     STA    STACK,6(A)      S-S'         (j+1,r)⇒栈
3H   ENT3   -1,5            S-S'+A''     置 r←j-1
     JMP    2B              S-S'+A''     转 Q2
8H   LD2    STACK,6(A)      S+1          Q8.弹出栈
     LD3    STACK,6(B)      S+1
     DEC6   1               S+1          (l,r)⇐栈
     J6NN   2B              S+1          如果栈不空则转 Q2
9H   ENT5   2-N             1            Q9.直接插入排序。j←2
2H   LDA    INPUT+N,5       N-1          K←K_j,R←R_j
     CMPA   INPUT+N-1,5     N-1          (在这个循环中,rI5≡j-N)
     JGE    6F              N-1          如果 K≥K_{j-1},则跳转
3H   ENT4   N-1,5           D            置 i←j-1
4H   LDX    INPUT,4         E
     STX    INPUT+1,4       E            置 R_{i+1}←R_i
```

```
        DEC4   1            E       置 i←i-1
        CMPA   INPUT,4      E
        JL     4B           E       如果 K<K_i,则重复
5H      STA    INPUT+1,4    D       R_{i+1}←R
6H      INC5   1            N-1
        J5NP   2B           N-1     2≤j≤N  ▌
```

“快速排序”的分析 用程序 Q 表示的计时信息,不难利用基尔霍夫守恒定律(参见 1.3.3 小节)和压入栈的每一个信息终归又被撤销这一事实导出。基尔霍夫定律应用于步骤 Q2 还表明

$$A = 1 + (S' + A''') + (S - S' + A'') + S = 2S + 1 + A'' + A''' \qquad (15)$$

因此总共的运行时间为

$$24A + 11B + 4C + 3D + 8E + 7N + 9S \text{ 单位}$$

其中

A = 分划阶段数;
B = 在步骤 Q6 中的交换数;
C = 在进行分划时所作比较数;
D = 在直接插入(步骤 Q9)期间 $K_{j-1} > K_j$ 的次数;
E = 被直接插入消去的反序数;
S = 某项压入栈的次数。 (16)

通过分析这 6 个量,就能对参数 M 进行精巧的选择,M 是用来确定直接插入和分划之间的“阈值”的。这个分析是特别有教益的,因为这个算法比较复杂;揭示这个复杂性是阐明重要技术的一个好方法。然而,非数学方面的读者,请跳到等式(25)。

像在这一章中大多数其它的分析那样,假定有待排序的这些键码是不同的;习题 18 指出,这些键码之间的相等性并不严重地损害算法 Q 的效率,而且事实上它们似乎还有助于提高效率。由于这个方法仅依赖于键码的相对次序,也可以假定,它们只不过是在某种次序下的$\{1,2,\cdots,N\}$。

我们可以通过考虑最初分划阶段的行为来着手解决这个问题,这个最初的阶段使我们头一次到 Q7 去。一旦已经实现了这个分划,则如果原来的文件是在随机次序下的,$R_1 \cdots R_{j-1}$和 $R_{j+1} \cdots R_N$ 两个子文件也将是在随机次序下,因为在这些子文件中元素的相对次序对分划算法没有影响。因此随后的分划的贡献可以通过对 N 用归纳法确定(这是一个重要的发现,因为某些违背这个性质的其它算法已经证明是相当慢的;见*Computing Surveys* **6** (1974),287~289)。

设 s 是头一个键码 K_1 的值,并假定 $K_1, \cdots K_s$ 中恰有 t 个大于 s(记住正被排序的键码是整数$\{1,2,\cdots,N\}$)。如果 $s=1$,则容易看出在分划的头一阶段发生了什么

情况:步骤 Q3 被执行一次,步骤 Q4 被执行 N 次,然后步骤 Q5 使我们转去执行 Q7。所以在此情况下头一个阶段的贡献是 $A=1,B=0,C=N+1$。当 $s>1$ 时(见习题 21),一个类似的但稍微复杂的论证说明,一般地,头一阶段对于总运行时间的贡献是

$$A = 1, B = t, C = N + 1 \qquad \text{对于 } 1 \leqslant s \leqslant N \tag{17}$$

对此,还要加上稍后阶段的贡献,这些阶段分别对具有 $s-1$ 和 $N-s$ 个元素的子文件进行排序。

如果假定原始的文件是随机次序的,则现在已有可能写下定义 $A,B,\cdots,S$ 的概率分布的生成函数公式(见习题 22)。但为了简单起见,这里仅仅考虑这些量作为 N 的函数的平均值 $A_N,B_N,\cdots,S_N$。例如,考虑在分划过程中出现的比较平均次数 C_N。当 $N\leqslant M$ 时,$C_N=0$。否则由于任意给定的 s 值以 $1/N$ 的概率出现,我们有

$$C_N = \frac{1}{N}\sum_{S=1}^{N}(N + 1 + C_{s-1} + C_{N-s}) =$$

$$N + 1 + \frac{2}{N}\sum_{0\leqslant k<N} C_k, \qquad \text{对于 } N > M \tag{18}$$

对于其它的量,A_N,B_N,D_N,E_N,S_N,类似的公式成立(见习题 23)。

有一个简单的方法来解形如

$$x_n = f_n + \frac{2}{n}\sum_{0\leqslant k<n} x_k, \quad \text{对于 } n \geqslant m \tag{19}$$

的递推关系。头一步是脱去求和式符号

$$(n+1)x_{n+1} = (n+1)f_{n+1} + 2\sum_{0\leqslant k\leqslant n} x_k$$

$$nx_n = nf_n + 2\sum_{0\leqslant k<n} x_k$$

可以把两者相减,得到

$$(n+1)x_{n+1} - nx_n = g_n + 2x_n, \quad \text{其中 } g_n = (n+1)f_{n+1} - nf_n$$

现在递推式有了简单得多的形式

$$(n+1)x_{n+1} = (n+2)x_n + g_n, \qquad \text{对于 } n \geqslant m \tag{20}$$

任何具有一般形式

$$a_n x_{n+1} = b_n x_n + g_n \tag{21}$$

的递推关系可以约化为一个和式,只要以"求和因子"$a_0a_1\cdots a_{n-1}/b_0b_1\cdots b_n$ 乘以两边即可;我们得到

$$y_{n+1} = y_n + c_n, \quad \text{其中 } y_n = \frac{a_0\cdots a_{n-1}}{b_0\cdots b_{n-1}}x_n, \quad c_n = \frac{a_0\cdots a_{n-1}}{b_0b_1\cdots b_n}g_n \tag{22}$$

在情况(20)中,求和因子就是 $n!/(n+2)!=1/(n+1)(n+2)$,所以我们发现简单的关系

$$\frac{x_{n+1}}{n+2}=\frac{x_n}{n+1}+\frac{(n+1)f_{n+1}-nf_n}{(n+1)(n+2)},\quad \text{对于 } n\geqslant m \tag{23}$$

是(19)的一个推论。

例如,如果我们置 $f_n=1/n$,则对于所有 $n\geqslant m$,得到意外的结果 $x_n/(n+1)=x_m/(m+1)$。如果我们置 $f_n=n+1$,则对于所有 $n\geqslant m$,得到

$$x_n/(n+1)=2/(n+1)+2/n+\cdots+2/(m+2)+x_m/(m+1)=$$
$$2(H_{n+1}-H_{m+1})+x_m/(m+1)$$

因此通过置 $m=M+1$ 和对于 $n\leqslant M$, $x_n=0$,我们得到(18)的解;所求的公式是

$$C_N=(N+1)(2H_{N+1}-2H_{M+2}+1)\approx$$
$$2(N+1)\ln\left(\frac{N+1}{M+2}\right),\qquad \text{对于 } N>M \tag{24}$$

习题 6.2.2-8 证明,当 $M=1$ 时,C_N 的标准差近似于$\sqrt{(21-2\pi^2)/3}N$;同(24)相比这是相当小的。

以类似的方式,可以求出其它的量(见习题 23);当 $N>M$ 时我们有

$$A_N=2(N+1)/(M+2)-1,$$
$$B_N=\frac{1}{6}(N+1)\left(2H_{N+1}-2H_{M+2}+1-\frac{6}{M+2}\right)+\frac{1}{2},$$
$$D_N=(N+1)(1-2H_{M+1}/(M+2)),$$
$$E_N=\frac{1}{6}(N+1)M(M-1)/(M+2),$$
$$S_N=(N+1)/(2M+3)-1,\quad \text{对于 } N>2M+1 \tag{25}$$

上面的讨论说明,通过使用以前仅仅应用于较简单情况的那些技术,有可能对相当复杂的程序的平均运行时间进行精确的分析。

公式(24)和(25)可以用来确定在一台具体的计算机上 M 的“最好”的值。在 MIX 的情况下,对于 $N>2M+1$,程序 Q 平均要求$(35/3)(N+1)H_{N+1}+\frac{1}{6}(N+1)f(M)-34.5$ 个单位的时间,其中

$$f(M)=8M-70H_{M+2}+71-36\frac{H_{M+1}}{M+2}+\frac{270}{M+2}+\frac{54}{2M+3} \tag{26}$$

我们要选择 M 使得 $f(M)$是一个极小值,而且简单的计算机计算表明 $M=9$ 是最好的。对于很大的 N,当 $M=9$ 时,程序 Q 的平均运行时间近似为 $11.667(N+1)\ln N-1.74N-18.74$ 个单位。

考虑到程序 Q 只要很少的存储空间,所以平均说来,它是十分快的。它的速度主要取决于以下事实,即在步骤 Q3 和 Q4 中的内循环是极其短的——每个仅含 3 条 MIX 指令(见行 12～14 和 15～17)。在步骤 Q6 中的交换次数,仅大约是步聚 Q3 和 Q4 中的比较数的 1/6,因此通过在内循环中不比较 i 和 j,我们已节省相当的时间。

但对于算法 Q,最坏情况是什么呢?是否有某些它不能有效处理的输入?对于这个问题的答案是十分令人难堪的:如果原来的文件已经是有序的,且 $K1<K2<\cdots<K_N$,则每个分划"操作"几乎都是无用的,因为它仅使子文件的大小减少一个元素!所以这种情况(它应是所有情况中最易于排序的)使得快速排序根本不快;它的排序时间同 N^2,而不是同 $N\log_2 N$ 成比例(见习题 25)。和已经见到的其它排序方法不同,算法 Q 偏爱一个无次序的文件!

Hoare 在他开创性的论文中,曾建议用两种方法来弥补这种情况,其原理是选择支配分划的测试键码 K 的较好值。他的建议之一是在步骤 Q2 的最后部分中选择 l 和 r 之间的一个随机整数 q;我们可以在该步中把指令"$K\leftarrow K_l$"改成为

$$K \leftarrow K_q,\quad R \leftarrow R_q,\quad R_q \leftarrow R_q,\quad R_l \leftarrow R \tag{27}$$

(最后的赋值"$R_l\leftarrow R$"是必要的;否则当 K 是被分划的子文件的最小键码时步骤 Q4 将以 $j=l-1$ 停止)。按照等式(25),这样的随机整数平均说来只需要被计算 $2(N+1)/(M+2)-1$ 次,所以附加的运行时间不是很多的;因此随机选择针对最坏情况的出现给出很好的保护。甚至一个温和地选择的 q 都将是安全的。习题 58 证明,通过真正随机的 q,比如说,多于 $20N/\ln N$ 次比较的概率将肯定地小于 10^{-8}。

Hoare 的第二个建议是考察文件的小样品,并选择这个样品的一个中值。这个方法已为 R. C. Singleton[*CACM* **12**(1969),185~187]所采用。他建议令 K_q 是 3 个值

$$K_l,\quad K_{\lfloor (l+r)/2\rfloor},\quad K_r \tag{28}$$

的中值。Singleton 过程把比较的次数从 $2N\ln N$ 减少到大约 $\frac{12}{7}N\ln N$(见习题 29)。在这种情况下可以证明 B_N 渐近于 $C_N/5$,而不是 $C_N/6$,所以这个中值方法稍微增加了花费在传送数据中的时间数量;因此总运行时间大约减少 8%(详细的分析见习题 56)。最坏的情况仍然是 N^2 阶的,但是这样缓慢的行为很少出现。

W. D. Frazer 和 A. C. Mckellar[*JACM* **17**(1970),496~507]已经建议取一个更大得多的由 2^k-1 个记录组成的样品,其中把 k 选择成使 $2^k\approx N/\ln N$。这个样品可以用通常的快速排序方法进行排序,然后通过对这个文件进行 k 次扫描把样品插入到剩下的记录当中(把它划分成 2^k 个子文件,以样品的元素为界)。最后对这些子文件排序。当 N 在一个实用的范围内时,这样一个"样品排序"过程所需要的平均比较次数,大体等同于在 Singleton 中值方法中的次数,但是当 $N\to\infty$ 时,它减少到渐近值 $N\lg N$。

把快速排序同其他的方案组合在一起,可得到在最坏情况下 $O(N\log N)$ 排序时间的绝对保证,连同平均说来快速的运行时间。例如 D. R. Musser[*Software Practice & Exper.* **27** (1997),983~993]已经建议对快速排序栈上的每一项加上一个"分划深度"的分量。如果发现任何子文件已经被划分比如说多于 $2\lg N$ 次,我们就可放弃算法 Q,并且转到算法 5.2.3H。内循环时间保持不变,所以平均的总运行时间几乎保持和以前一样。

Robert Sedgewick 在*Acta Informatica* **7**(1977),327～356 上,以及在 *CACM* **21** (**1978**)**847～857**,**22** (1979),368 上对一些快速排序的优化变形做了分析。关于基于 15 年以上的经验,已被调整成适合于 UNIX 软件库的一个快速排序的版本,请参见 *J.L.Bentley* 和 *M.D.McIlroy* 的 *Software Practice & Exper* **23**(1993),1249～1265。

基数交换 现在讨论另一种方法,它非常不同于以前所看到的任何排序方案;它利用键码的二进制表示,所以仅仅适用于二进制计算机。这个方案不是比较两个键码,而是检查这些键码个别的二进位,看看它们是 0 还是 1。在其他方面,它有交换排序的特性,而且事实上,它颇类似于快速排序。由于它依赖于基数 2 的表示,故我们称它"基数交换排序"。这个算法可粗略描述如下:

1)按键码的最高二进位对序列排序,使得有前导 0 的所有键码都出现在有前导 1 的所有键码之前。这个排序首先寻找有前导 1 的最左边的键码 K_i,以及有前导 0 的最右边的键码 K_j;然后交换 R_i 和 R_j 并重复这个过程,直到 $i>j$。

2)设 F_0 是具有前导 0 的诸元素,F_1 是其余的元素。对 F_0 应用基数交换排序方法(现在从左边第二位开始而不是从最高位开始),直到 F_0 整个地被排序为止;然后对 F_1 同样这样做。

例如,表 3 示出了对于我们的 16 个随机数如何进行基数交换排序,这些数已被转换为八进制。表中的阶段 1 示出初始的输入,在对第 1 位进行交换之后,我们到达阶段 2。阶段 2 按第 2 位对头一组排序,而阶段 3 按第 3 位进行工作(读者可用心算把八进转换成 10 位二进制数,例如,0232 代表$(0\ 010\ 011\ 010)_2$)。当按第 4 位进行排序后到达阶段 5 时,发现剩下的每一组只有一个元素,所以文件的这部分已不必进一步考察。"^{4}[0232 0252]"指的是下一步要按左起第 4 位对子文件 0232 0252 进行排序。对本例来说,当按第 4 位进行排序时,没有进展;需要进到第 5 位,才能把这些项区分开来。

表 3 中所示的整个排序过程花费 22 个阶段,稍多于快速排序的相应数目(表 2)。类似地,按位检查的数目为 82 次,是相当高的;但将看到,假定这些键码是一致分布的,则对于很大的 N 来说,按位检查的数目实际上小于快速排序所做的比较数目。表 3 中的交换总数是 17 次,它是完全适度的。注意,尽管被排序的是 10 位数,但这里按位检查决没有超过第 7 位。

如同在快速排序中那样,可以使用一个栈来记住正在等候的子文件的"边界行信息"。一种方便的办法是不首先去对最小的子文件排序,而是简单地从左到右进行排序,因为在这种情况下,栈的大小决不能超过正被排序的键码的位数。在下列算法中,栈项目(r,b)用来表示将被按第 b 位进行排序的一个子文件的右边界;左边界不必记录在这个栈中,由于这一过程的自左到右的本性,它是隐含的。

算法 R (基数交换排序) 适当地重新排列记录 $R_1,\cdots,R_N$;在排序完成之后,它们的键码将是有序的,$K_1\leqslant\cdots\leqslant K_N$。假定每个键码都是一个非负的 m 位的二进

表 3 基数交换排序

阶段																	*l*	*r*	*b*	栈
1	^{1}⟦0767	0127	1000	0075	1614	0252	1601	0423	1215	0652	0232	0775	1144	1245	1375	1277⟧	1	16	1	—
2	^{2}⟦0767	0127	0775	0075	0232	0252	0652	0423⟧^{2}	[1215	1601	1614	1000	1144	1245	1375	1277]	1	8	2	(16,2)
3	^{3}⟦0252	0127	0232	0075⟧	^{3}[0775	0767	0652	0423]	^{2}[1215	1601	1614	1000	1144	1245	1375	1277]	1	4	3	(8,3)(16,2)
4	^{4}⟦0075	0127⟧	^{4}[0232	0252]	^{3}[0775	0767	0652	0423]	^{2}[1215	1601	1614	1000	1144	1245	1375	1277]	1	2	4	(4,4)(8,3)(16,2)
5	0075	0127	^{4}⟦0232	0252⟧	^{3}[0775	0767	0652	0423]	^{2}[1215	1601	1614	1000	1144	1245	1375	1277]	3	4	4	(8,3)(16,2)
6	0075	0127	^{5}⟦0232	0252⟧	^{3}[0775	0767	0652	0423]	^{2}[1215	1601	1614	1000	1144	1245	1375	1277]	3	4	5	(8,3)(16,2)
7	0075	0127	0232	0253	^{3}⟦0775	0767	0652	0423⟧	^{2}[1215	1601	1614	1000	1144	1245	1375	1277]	5	8	3	(16,2)
8	0075	0127	0232	0252	0423	^{4}⟦0767	0652	0775⟧	^{2}[1215	1601	1614	1000	1144	1245	1375	1277]	6	8	4	(16,2)
9	0075	0127	0232	0252	0423	0652	^{5}⟦0767	0775⟧	^{2}[1215	1601	1614	1000	1144	1245	1375	1277]	7	8	5	(16,2)
10	0075	0127	0232	0252	0423	0652	^{6}⟦0767	0775⟧	^{2}[1215	1601	1614	1000	1144	1245	1375	1227]	7	8	6	(16,2)
11	0075	0127	0232	0252	0423	0652	^{7}⟦0767	0775⟧	^{2}[1215	1601	1614	1000	1144	1245	1375	1277]	7	8	7	(16,2)
12	0075	0127	0232	0252	0423	0652	0767	0075	^{2}[1215	1601	1614	1000	1144	1245	1375	1277]	9	16	2	—
13	0075	0127	0232	0252	0423	0652	0767	0775	^{3}⟦1215	1277	1375	1000	1144	1245⟧	^{3}[1614	1601⟧	9	14	3	(16,3)
14	0075	0127	0232	0252	0423	0652	0767	0775	^{4}⟦1144	1000⟧	^{4}[1375	1277	1215	1245]	^{3}[1614	1601]	9	10	4	(14,4)(16,3)
15	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	^{4}⟦1375	1277	1215	1245⟧	^{3}[1614	1601]	11	14	4	(16,3)
16	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	^{5}⟦1245	1277	1215⟧	^{5}[1375]	^{3}[1614	1601]	11	13	5	(14,5)(16,3)
17	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	1215	^{6}⟦1277	1245⟧	^{5}[1375]	^{3}[1614	1601]	12	13	6	(14,5)(16,3)
18	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	1215	1245	1277	1375	^{3}⟦1614	1601⟧	15	16	3	—
19	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	1215	1245	1277	1375	^{4}⟦1614	1601⟧	15	16	4	—
20	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	1215	1245	1277	1375	^{5}⟦1614	1601⟧	15	16	5	—
21	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	1215	1245	1277	1375	^{6}⟦1614	1601⟧	15	16	6	—
22	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	1215	1245	1277	1375	^{7}⟦1614	1601⟧	15	16	7	
23	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	1215	1245	1277	1375	1601	1614	17	—	—	—

注:基数交换方法精确地考察每一个这样的二进位恰一次,它是为确定键码的最终次序所需要的。

数,$(a_1a_2\cdots a_m)_2$;第 i 个最高位 a_i,称为这个键码的"位 i"。需要一个至多能存下 $m-1$ 个项目的辅助栈做临时存储。本算法实质上遵循上文中所描述的基数交换分划过程;对它的效率可以有若干改进,如下文和习题中所描述的那样。

R1. [初始化] 置栈为空,并置 $l\leftarrow 1, r\leftarrow N, b\leftarrow 1$。

R2. [开始新阶段] (现在希望按位 b 对子文件 $R_l\leqslant\cdots\leqslant R_r$ 排序;由于这个算法的本性,我们有 $l\leqslant r$) 如果 $l=r$,则转到步骤 R10(由于一个单字文件已被排好序)。否则置 $i\leftarrow l, j\leftarrow r$。

R3. [检查 K_i 中的 1] 考察 K_i 的位 b。如果它是 1,则转到步骤 R6。

R4. [i 增值] i 增加 1。如果 $i\leqslant j$,则返回步骤 R3;否则,转到步骤 R8。

R5. [检查 K_{j+1} 中的 0] 考察 K_{j+1} 的位 b。如果它为 0,则转到步骤 R7。

R6. [j 减值] j 减 1。如果 $i\leqslant j$,则转到步骤 R5;否则,转到步骤 R8。

R7. [交换 R_i, R_{j+1}] 交换记录 $R_i\leftrightarrow R_{j+1}$;然后转到步骤 R4。

R8. [检查特殊情况] (这时已经完成了一个分划阶段;$i=j+1$,键码 $K_l,\cdots,K_j$ 的位 b 是 0,而键码 $K_i,\cdots,K_r$ 的位 b 是 1) b 增加 1。如果 $b>m$,其中 m 是在这些键码中总的位数,则转到步骤 R10(在这样一种情况下,子文件 $R_l\cdots R_r$ 已被排好序。如果在文件中不可能出现相等的键码,则不必进行这个检查)。否则,如果 $j<l$ 或 $j=r$,则返回步骤 R2(所有考察的位都分别为 1 或都为 0)。否则,如果 $j=l$,则 j 增加 1 并且转到步骤 R2(仅有一个位为 0)。

R9. [压入栈] 在栈顶插入项目(r,b),然后置 $r\leftarrow j$ 并转到步骤 R2。

R10. [弹出栈] 如果栈是空的,则已经完成了排序;否则置 $l\leftarrow r+1$,删取栈顶的项目(r',b'),置 $r\leftarrow r', b\leftarrow b'$,并返回步骤 R2。 ▌

程序 R(基数交换排序) 下列 MIX 代码基本上使用和程序 Q 同样的约定。我们有 $rI1\equiv l-r$, $rI2\equiv r$, $rI3\equiv i$, $rI4\equiv j$, $rI5\equiv m-b$, rI6≡栈的大小,但对于某些(下边指出的)指令保持 $rI3\equiv i-j$ 或 $rI4\equiv j-i$ 是方便的。由于基数交换的二进制属性,这个程序使用操作 SRB(二进制右移 AX),JAE(A 为偶数时转移)以及 JAO(A 为奇数时转移),这些操作已在 4.5.2 小节中定义。假定 $N\geqslant 2$。

```
START  ENT60  1                      R1.初始化。置栈为空
       ENT1   1-N          1         l←1
       ENT2   N            1         r←N
       ENT5   M-1          1         b←1
       JMP    1F           1         转 R2(省略检验 l=r)
9H     INC6   1            S         R9.压入栈。[rI4=l-j]
       ST2    STACK,6(A)   S
       ST5    STACK,6(B)   S         (r,b)⇒栈
```

	ENN1	0,4	S	$rI1 \leftarrow l - j$
	ENT2	−1,3	S	$r \leftarrow j$
1H	ENT3	0,1	A	*R2. 开始新阶段* $[rI3 = i - j]$
	ENT4	0,2	A	$i \leftarrow l, j \leftarrow r$ $[rI3 = i - j]$
3H	INC3	0,4	C'	*R3. 检查 K_i 是否为 1*
	LDA	INPUT,3	C'	
	SRB	0,5	C'	rA 的个位←K_i 的位 b
	JAE	4F	C'	如果它为 0,则转到 R4
6H	DEC4	1,3	$C'' + X$	*R6. j 减值。* $j \leftarrow j - 1$
				$[rI4 = j - i]$
	J4N	8F	$C'' + X$	如果 $j < i$ 则转到 R8
				$[rI4 = j - i]$
5H	INC4	0,3	C''	*R5. 检查 K_{j+1} 是否为 0*
	LDA	INPUT+1,4	C''	
	SRB	0,5	C''	rA 的个位←K_{j+1} 的位 b
	JAO	6B	C''	如果它为 1,则转到 R6
7H	LDA	INPCT+1,4	B	*R7. 交换 R_i, R_{i+1}*
	LDX	INPUT,3	B	
	STX	INPUT+1,4	B	
	STA	INPUT,3	B	
4H	DEC3	−1,4	$C' - X$	*R4. i 增值。* $i \leftarrow i + 1$
				$[rI3 = i - j]$
	J3NP	3B	$C' - X$	如果 $i \leqslant j$ 则转 R3
				$[rI3 = i - j]$
	INC3	0,4	$A - X$	$rI3 \leftarrow i$
8H	J5Z	0F	A	*R8. 检验特殊情况*
				[rI4 = 未知]
	DEC5	1	$A - G$	如果 $b = m$ 则转到 R10,否则 $b \leftarrow b - 1$
	ENT4	−1,3	$A - G$	$rI4 \leftarrow j$
	DEC4	0,2	$A - G$	$rI4 \leftarrow j - r$
	J4Z	1B	$A - G$	如果 $j = r$,则转到 R2
	DEC4	0,1	$A - G - R$	$rI4 \leftarrow j - l$

```
        J4N     1B              A−G−R     如果 j<l,则转到 R2
        J4NZ    9B              A−G−L−R   如果 j≠l,则转到 R9
        INC1    1               K         l←l+1
2H      J1NZ    1B              K+S       如果 l≠r,则转移
0H      ENT1    1,2             S+1       R10.弹出栈
        LD2     STACK,6(A)      S+1
        DEC1    0,2             S+1
        LD5     STACK,6(B)      S+1       栈⇒(r,b)
        DEC6    1               S+1
        J6NN    2B              S+1       如果栈非空,则转到 R2  ▌
```

这个基数交换程序的运行时间依赖于:

A = 遇到的 $l<r$ 的阶段数;

B = 交换数;

$C=C'+C''$ = 位检查的数目;

G = 在步骤 R8 中 $b>m$ 的次数;

K = 在步骤 R8 中 $b\leqslant m, j=l$ 的次数;

L = 在步骤 R8 中 $b\leqslant m, j<l$ 的次数;

R = 在步骤 R8 中 $b\leqslant m, j=r$ 的次数;

S = 元素压入栈的次数;

X = 在步骤 R6 中 $j<i$ 的次数。 (29)

由基尔霍夫定律,$S=A-G-K-L-R$;所以总运行时间为 $27A+8B+8C-23G-14K-17L-19R-X+13$ 个单位。如习题 34 所示,以一个较复杂的程序为代价,位检查循环可以进行得更快些。每当 $r-l$ 充分小时,像在算法 Q 中那样,也可通过使用直接插入来提高基数交换的速度;但此处我们并不详细介绍这些精化措施。

为了分析基数交换的运行时间,可以使用两种类型的输入数据:

(i)假定 $N=2^m$ 并且待排序的键码就是随机顺序下的整数 $0,1,2,\cdots,2^m-1$;

(ii)假定 $m=\infty$(无限地精确)而且有待排序的键码是在[0..1)中独立地一致分布的实数。

情况(i)的分析相对来讲是容易的,所以已留作读者的习题(见习题 35)。情况(ii)相对来讲是困难的,所以它也留作一个习题(见习题 38)。下面为对于这些分析结果的粗略近似:

量	情况(i)	情况(ii)
A	N	αN

B	$\frac{1}{4}N\lg N$	$\frac{1}{4}N\lg N$
C	$N\lg N$	$N\lg N$
G	$\frac{1}{2}N$	0
K	0	$\frac{1}{2}N$
L	0	$\frac{1}{2}(\alpha-1)N$
R	0	$\frac{1}{2}(\alpha-1)N$
S	$\frac{1}{2}N$	$\frac{1}{2}N$
X	$\frac{1}{2}N$	$\frac{1}{4}(\alpha+1)N$ (30)

这里 $\alpha=1/\ln 2\approx1.4427$。注意尽管情况(ii)花费的阶段要多 44%,但是交换、位检查和栈访问的平均次数,对于两种情况来说实质上是相同的。平均来说,我们的 MIX 程序,在情况(ii)下对 N 个项目排序花费了近$14.4N\ln N$个时间单位,而且使用习题 34 的建议还可以减少到大约 $11.5N\ln N$;对于程序 Q,对应的数字是 $11.7N\ln N$,使用 Singleton 三个取中的建议,它可以减少到大约 $10.6N\ln N$。

于是,当对一致分布的数据排序时,平均说来,基数交换排序和快速排序花费大约同样长的时间。在某些机器上,它实际上比快速排序还稍微更快些。习题 53 指出,对于非一致分布的数据这个过程减慢到什么程度。重要的是要注意,整个分析是以所有键码都不相同这个假定为依据的;*当出现相等的键码时,如上定义的基数交换就不是特别有效的*。因为在 b 变成 $>m$ 之前,它白白浪费一些阶段来分开相等的键码的集合。弥补这个缺陷的一个似乎有理的方法,在习题 40 的答案中提出。

基数交换和快速排序两者实质上都是以分划的思想为基础的。诸记录被交换,直到这文件被分成两个部分为止:一个左边的子文件,其中所有的键码都$\leq K$(对某个 K),以及一个右边的子文件,其中所有的键码都$\geq K$。快速排序把这个文件中的一个实际的键码选为 K,而基数交换则实质上以二进表示为基础选择一个人为的键码 K。从历史上看,基数交换是由 P. Hildebrandt, H. Isbitz, H. Rising 以及 J. Schwartz[*JACM* **6**(1959),156～163]发现的,大约比快速排序早一年。其它的分划方案也是可能的;例如,当已知所有的键码位于 u 和 v 之间时,Johy McCarthy 建议置 $K\approx\frac{1}{2}(u+v)$。Yihsiao Wang(王义孝)已经提议把诸如(28)的 3 个键码值的均值用作分划的阈值;他证明了对一致分布的随机数据排序,所需要的比较次数近似为 $1.082N\lg N$。

M.H.van Emden 提出了另外一种分划策略[*CACM* **13**(1970),563~567]:他不是预先选择 K,而是随着分划的进行,记住 $K'=\max(K_l,\cdots,K_i)$和 $K''=\min(K_j,\cdots,K_r)$,“学会”如何选一个好的 K。我们可以令 i 增值,直到遇到一个大于 K'的键码为止,然后令 j 减值,直到遇到一个小于 K''的键码为止,然后交换和/或调整 K'和 K''。对于这个“区间交换排序”方法的经验测试指出,它运行得比快速排序稍微慢些,它的运行时间看起来是这样难以分析,以致于绝对找不到一个适当的理论解释,特别是因为在分划之后诸文件已不再处于随机次序了。

把基数交换推广到大于 2 的基数问题,在 5.2.5 小节中讨论。

***渐近方法** 交换排序算法的分析导致了某些特别有教益的数学问题,这些问题使我们更多地学会如何研究诸函数的渐近行为。例如,在对冒泡排序的分析中(等式 9)遇到过函数

$$W_n = \frac{1}{n!}\sum_{0\leqslant r<s\leqslant n} s!\,r^{n-s} \tag{31}$$

它的渐近值是什么呢?

我们可以照搬对对合数等式 5.1.4-(41)的研究方式;读者将发现在进一步阅读之前,复习一下 5.1.4 小节末尾的讨论是有帮助的。

检查(31)可知,$s=n$ 的贡献大于 $s=n-1$,等等;这提示以 $n-s$ 代替 s。事实上,立即发现,使用替换 $t=n-s+1$,$m=n+1$ 把(31)变为

$$\frac{1}{m}W_{m-1} = \frac{1}{m!}\sum_{1\leqslant t<m}(m-t)!\sum_{0\leqslant r<m-t} r^{t-1} \tag{32}$$

是最方便的。内层的求和可由欧拉求和公式得到熟知的渐近级数,即

$$\sum_{0\leqslant r<N} r^{t-1} = \frac{N^t}{t} - \frac{1}{2}(N^{t-1}-\delta_{t1}) + \frac{B_2}{2!}(t-1)(N^{t-2}-\delta_{t2}) + \cdots = \frac{1}{t}\sum_{j=0}^{k}\binom{t}{j}B_j(N^{t-j}-\delta_{tj}) + O(N^{t-k}) \tag{33}$$

(见习题 1.2.11.2-4),因此问题归结为研究形如

$$\frac{1}{m!}\sum_{1\leqslant t<m}(m-t)!(m-t)^t t^k,\ k\geqslant -1 \tag{34}$$

的和。如同在 5.1.4 小节那样,可以证明每当 t 大于 $m^{1/2+\epsilon}$时,这个求和数的值是可以忽略的 $O(\exp(-n^\delta))$,因此可以置 $t=O(m^{1/2+\epsilon})$,而且用斯特林近似代替这些阶乘:

$$\frac{(m-t)!(m-t)^t}{m!} = \sqrt{1-\frac{t}{m}}\exp\left(\frac{t}{12m^2} - \left(\frac{t^2}{2m}+\frac{t^3}{3m^2}+\frac{t^4}{4m^3}+\frac{t^5}{5m^4}\right) + O(m^{-2+6\epsilon})\right)$$

我们因此对

$$r_k(m) = \sum_{1\leqslant t<m} e^{-t^2/2m}t^k,\quad k\geqslant -1 \tag{35}$$

的渐近值感兴趣。这个和也可以扩展到整个 $1\leqslant t<\infty$的范围而不会改变它的渐近

值,因为对于 $t > m^{1/2+\epsilon}$ 的那些值是可以忽略的。

设 $g_k(x) = x^k e^{-x^2}$ 和 $f_k(x) = g_k(x/\sqrt{2m})$。当 $k \geqslant 0$ 时,欧拉求和公式告诉我们

$$\sum_{0 \leqslant t < m} f_k(t) = \int_0^m f_k(x)\mathrm{d}x + \sum_{j=1}^{p} \frac{B_j}{j!}(f_k^{(j-1)}(m) - f_k^{(j-1)}(0)) + R_p$$

$$R_p = \frac{(-1)^{p+1}}{p!}\int_0^m B_p(\{x\})f_k^{(p)}(x)\mathrm{d}x = \left(\frac{1}{\sqrt{2m}}\right)^p O\left(\int_0^\infty |g_k^{(p)}(y)|\mathrm{d}y\right) = O(m^{-p/2}) \tag{36}$$

因此每当 $k \geqslant 0$ 时,通过使用实质上和以前使用过的同样的思想,就能得到一个 $r_k(m)$ 的渐近级数。但当 $k = -1$ 时,这个方法就失灵了,因为 $f_{-1}(0)$ 无定义;我们也不能仅仅从 1 到 m 求和,因为当下限为 1 时,余数不给出越来越小的 m 的乘方(这是问题的症结,因而读者在进一步阅读之前应确保自己了解这个问题)。

为了摆脱这个困境,可以定义 $g_{-1}(x) = (e^{-x^2} - 1)/x$, $f_{-1}(x) = g_{-1}(x/\sqrt{2m})$;然后可以以简单的方式从 $\Sigma_{0 \leqslant t < m} f_{-1}(t)$ 得到 $f_{-1}(0) = 0$ 和 $r_{-1}(m)$。等式(36)现在对 $k = -1$ 成立,而且剩下的积分是"熟知的",由习题 43,

$$\frac{2}{\sqrt{2m}}\int_0^m f_{-1}(x)\mathrm{d}x = 2\int_0^m \frac{e^{-x^2/2m} - 1}{x}\mathrm{d}x = \int_0^{m/2} \frac{e^{-y} - 1}{y}\mathrm{d}y = \int_0^1 \frac{e^{-y} - 1}{y}\mathrm{d}y + \int_1^{m/2} \frac{e^{-y}}{y}\mathrm{d}y - \ln\frac{m}{2} = -\gamma - \ln m + \ln 2 + O(e^{-m/2}),$$

现在已经有了足够的事实和公式来作出答案,如习题 44 所示

$$W_n = \frac{1}{2}m\ln m + \frac{1}{2}(\gamma + \ln 2)m - \frac{2}{3}\sqrt{2\pi m} + \frac{49}{36} + O(n^{-1/2}), \quad m = n + 1 \tag{37}$$

这就完成了对于冒泡排序的分析。

对于基数交换排序的分析,需要知道当 $n \to \infty$ 时有限和

$$U_n = \sum_{k \geqslant 2}\binom{n}{k}(-1)^k \frac{1}{2^{k-1} - 1} \tag{38}$$

的渐近值。这个问题要比至今遇到的任何其它渐近问题更难些;幂级数展开,欧拉求和公式等的初等方法,证明是不适用的。N. G. de Bruijn 提出了下列推导。

为了摆脱在(38)中大因子 $\binom{n}{k}(-1)^k$ 的抵消作用,我们把和重新写成一个无穷级数

$$U_n = \sum_{k \geqslant 2}\binom{n}{k}(-1)^k \sum_{j \geqslant 1}\left(\frac{1}{2^{k-1}}\right)^j =$$

$$\sum_{j\geqslant 1}(2^j(1-2^{-j})^n-2^j+n) \tag{39}$$

如果置 $x=n/2^j$,求和的数是

$$2^j(1-2^{-j})^n-2^j+n=\frac{n}{x}\left(\left(1-\frac{x}{n}\right)^n-1+x\right)$$

当 $x\leqslant n^\epsilon$ 时,有

$$\left(1-\frac{x}{n}\right)^n=\exp\left(n\ \ln\left(1-\frac{x}{n}\right)\right)=\exp(-x+x^2O(n^{-1})) \tag{40}$$

而这提示以

$$T_n=\sum_{j\geqslant 1}(2^je^{-n/2^j}-2^j+n) \tag{41}$$

逼近(39)。为了论证这个逼近,有 $U_n-T_n=X_n+Y_n$,其中

$$X_n=\sum_{\substack{j\geqslant 1\\2^j<n^{1-\epsilon}}}(2^j(1-2^{-j})^n-2^je^{-n/2^j})= \qquad [\text{即 } x>n^\epsilon \text{ 的诸项}]$$

$$\sum_{\substack{j\geqslant 1\\2^j<n^{1-\epsilon}}}O(ne^{-n/2^j})= \qquad [\text{因为 } 0<1-2^{-j}<e^{-2^{-j}}]$$

$$O(n\log n e^{-n^\epsilon}) \qquad [\text{因为有 } O(\log n) \text{ 项}]$$

以及

$$Y_n=\sum_{\substack{j\geqslant 1\\2^j\geqslant n^{1-\epsilon}}}(2^j(1-2^{-j})^n-2^je^{-n/2^j})= \qquad [x\leqslant n^\epsilon \text{ 的诸项}]$$

$$\sum_{\substack{j\geqslant 1\\2^j\geqslant n^{1-\epsilon}}}\left(e^{-n/2^j}\left(\frac{n}{2^j}\right)O(1)\right) \qquad [\text{由(40)}]$$

以下的讨论将揭示,后一个和是 $O(1)$;因此 $U_n-T_n=O(1)$(见习题 47)。

至今还未曾应用实际上不同于以前使用过的任何技术;但是 T_n 的研究需要一种以复变理论的简单原理为基础的新思想。如果 x 是任何正数,则有

$$e^{-x}=\frac{1}{2\pi i}\int_{1/2-i\infty}^{1/2+i\infty}\Gamma(z)x^{-z}dz=\frac{1}{2\pi}\int_{-\infty}^{\infty}\Gamma\left(\frac{1}{2}+it\right)x^{-(1/2+it)}dt \tag{42}$$

为了证明这个恒等式,考虑图 20(a)中所示的积分路径,其中 N,N' 和 M 是很大的。沿着这条回路的积分值是内部残数的和,即

$$\sum_{0\leqslant k<M}x^{-(-k)}\lim_{z\to -k}(z+k)\Gamma(z)=\sum_{0\leqslant k<M}x^k\frac{(-1)^k}{k!}$$

在顶线上的积分是 $O\left(\int_{-\infty}^{1/2}|\Gamma(t+iN)|x^{-t}dt\right)$,而且我们有熟知的界限

$$\Gamma(t+iN)=O(|t+iN|^{t-1/2}e^{-t-\pi N/2}) \qquad \text{当 } N\to\infty \text{ 时}$$

[关于伽玛函数的性质可见,例如 Erdélyi Magnus, Oberhettinger 以及 Tricomi 的,(*Higher Transcendental Functions* 1)(NewYork: McGraw - Hill, 1953),第 1 章]。因

此顶线积分是完全可以忽略的。$O\left(e^{-\pi N/2}\int_{-\infty}^{1/2}(N/xe)^t dt\right)$。底线积分也有一种类似的无关痛痒的行为。对于沿着左边线的积分，利用下列事实

$$\Gamma\left(\frac{1}{2}+it-M\right)=\Gamma\left(\frac{1}{2}+it\right)\Big/\left(-M+\frac{1}{2}+it\right)\cdots\left(-1+\frac{1}{2}+it\right)=$$
$$\Gamma\left(\frac{1}{2}+it\right)O\left(1/(M-1)!\right)$$

因此左边的积分是 $O(x^{M-1/2}/(M-1)!)\int_{-\infty}^{\infty}\left|\Gamma\left(\frac{1}{2}+it\right)\right|dt$。因而当 $M,N,N'\to\infty$ 时，仅仅右边的积分残存，而这就证明了(42)。事实上，如果以任何正数代替$\frac{1}{2}$，则(42) 成立。

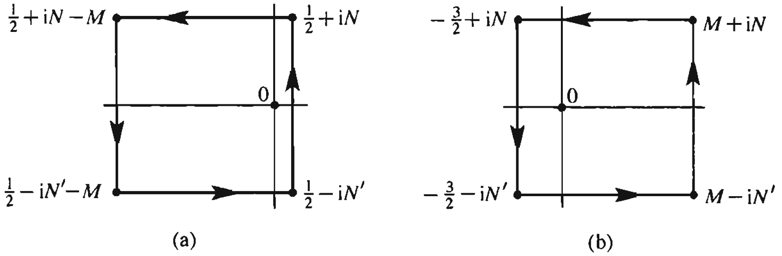


图 20 对于伽玛函数恒等式的积分回路

同样的论证可用来推导涉及伽玛函数的许多其它有用的关系。我们可以以 z 的其它函数代替 x^{-z}；或者可以用其它的量代替常数$\frac{1}{2}$。例如

$$\frac{1}{2\pi i}\int_{-3/2-i\infty}^{-3/2+i\infty}\Gamma(z)x^{-z}dz = e^{-x}-1+x \tag{43}$$

这是在 T_n 公式(41)中关键性的量:

$$T_n = n\sum_{j\geqslant 1}\frac{1}{2\pi i}\int_{-3/2-i\infty}^{-3/2+i\infty}\Gamma(z)(n/2^j)^{-1-z}dz \tag{44}$$

这个和可以放置到积分里边，因为收敛性是绝对好的特性。我们有

$$\sum_{j\geqslant 1}(n/2^j)^w = n^w\sum_{j\geqslant 1}(1/2^w)^j = n^w/(2^w-1)\text{，当 }\Re(w)>0\text{ 时}$$

因为$|2^w|=2^{\Re(w)}>1$，因此

$$T_n = \frac{n}{2\pi i}\int_{-3/2-i\infty}^{-3/2+i\infty}\frac{\Gamma(z)n^{-1-z}}{2^{-1-z}-1}dz \tag{45}$$

剩下的是计算后边的积分。

这次沿着远向右边扩展的一条路径进行积分，如图 20(b)所示。如果 $2^{iN}\neq 1$，

则顶线积分是 $O\left(n^{1/2}e^{-\pi N/2}\int_{-3/2}^{M} | M + iN |^t dt\right)$。而当 N 和N' 比M 大得多时，底线积分同样是可以忽略的。右线积分是 $O\left(n^{-1-M}\int_{-\infty}^{\infty} | \Gamma(M + it) | dt\right)$。固定 M 并命 $N, N' \to \infty$，证明 $-T_n/n$ 是$O(n^{-1-M})$加上在 $-3/2 < \Re(z) < M$ 范围内的残数之和。$T(z)$在 $z = -1$ 和 $z = 0$ 处有单极点，而 n^{-1-z}没有极点，而且当 $z = -1 + 2\pi ik/\ln 2$ 时 $1/(2^{-1-z} - 1)$有单极点。

在 $z = -1$ 处的双重极点是最难处理的。可以使用熟知的关系

$$\Gamma(z + 1) = \exp(-\gamma z + \zeta(2)z^2/2 - \zeta(3)z^3/3 + \zeta(4)z^4/4 - \cdots)$$

其中 $\zeta(s) = 1^{-s} + 2^{-s} + 3^{-s} + \cdots = H_\infty^{(s)}$，来导出当 $w = z + 1$ 很小时的下列展开式：

$$\Gamma(z) = \frac{\Gamma(w + 1)}{w(w - 1)} = -w^{-1} + (\gamma - 1) + O(w)$$

$$n^{-1-z} = 1 - w \ln n + O(w^2)$$

$$1/(2^{-1-z} - 1) = -w^{-1}/\ln 2 - \frac{1}{2} + O(w)$$

在 $z = -1$ 处的残数是这 3 个公式的乘积中 w^{-1}的系数，即$\frac{1}{2} - (\ln n + \gamma - 1)/\ln 2$。加上其它残数即得公式

$$\frac{T_n}{n} = \frac{\ln n + \gamma - 1}{\ln 2} - \frac{1}{2} + \delta(n) + \frac{2}{n} + O(n^{-M}) \tag{46}$$

其中，对任意大的 M，$\delta(n)$是颇奇怪的函数

$$\delta(n) = \frac{2}{\ln 2}\sum_{k\geqslant 1}\Re(\Gamma(-1 - 2\pi ik/\ln 2)\exp(2\pi ik \lg n)) \tag{47}$$

注意，$\delta(n) = \delta(2n)$。$\delta(n)$的平均值为 0，因为每项的平均值为 0（鉴于 4.2.4 小节中关于浮点数的结果，我们可以假定$(\lg n) \bmod 1$ 是一致分布的）。而且由于 $|\Gamma(-1 + it)| = |\pi/(t(1 + t^2)\sinh \pi t)|^{1/2}$，不难证明

$$|\delta(n)| < 0.000000173 \tag{48}$$

于是对于实际应用，可以放心地忽略 $\delta(n)$。然而在理论上，U_n 的渐近展开不能没有它；这就是为什么 U_n 是比较难分析的函数。

由(41)中 T_n 的定义，立即可以看出

$$\frac{T_{2n}}{2n} = \frac{T_n}{n} + 1 - \frac{1}{n} + \frac{e^{-n}}{n} \tag{49}$$

因此(46)中的误差项 $O(n^{-M})$是必不可少的；它不能以 0 代替。然而，习题 54 介绍了另一个分析方法，它通过推导出一个稍微奇特的收敛级数而避免这样的误差项。

总之，我们已经推导出困难的和(38)的特性：

$$U_n = n\lg n + n\left(\frac{\gamma - 1}{\ln 2} - \frac{1}{2} + \delta(n)\right) + O(1) \tag{50}$$

我们用来得到这个结果的伽玛函数是 Mellin 转换的一般技术的一个特殊情况。

Mellin 转换技术在研究面向基数的递推关系中极为有用。在习题 51～53 和 6.3 节中可以找到这个方法的其它例子。P. Flajolet, X. Gourdon 和 P. Dumas 在 *Theoretical Computer Science* **144**(1995), 3～58 中, 给出了对于 Mellin 转换及其在算法分析中的应用的一个精彩介绍。

习　题

1.[*M20*] 设 $a_1 \cdots a_n$ 是 $\{1, \cdots, n\}$ 的一个排列, 并设 i 和 j 是使得 $i<j$ 和 $a_i>a_j$ 的下标。令 $a_1' \cdots a_n'$ 是由 $a_1 \cdots a_n$ 通过交换 a_i 和 a_j 得到的排列。问 $a_1' \cdots a_n'$ 能有比 $a_1 \cdots a_n$ 更多的反序吗?

▶2.[*M25*] (a)对排列 3 7 6 9 8 1 4 5 2 进行排序的极小交换次数是多少? (b)一般地说, 给定 $\{1, \cdots, n\}$ 的任何排列 $\pi = a_1 \cdots a_n$, 设 $\mathrm{xch}(\pi)$ 是把 π 排序成为递增次序的极小交换次数。试借助于 π 的"较简单的"特征来表达 $\mathrm{xch}(\pi)$(有关测量一个排列的无序的另外一个方法, 参见习题 5.1.4 -41)。

3.[10] 冒泡排序算法 B 是一个稳定的排序算法吗?

4.[*M23*] 如果在步骤 B4 中 $t=1$, 则实际上可以立即终止算法 B, 因为随后的步骤 B2 已没有什么有用的事要做。当对一个随机排列排序时, 在步骤 B4 中将出现 $t=1$ 的概率是多少?

5.[*M25*] 设 $b_1 b_2 \cdots b_n$ 是排列 $a_1 a_2, \cdots, a_n$ 的反序表。试证, 对于 $0 \leqslant r \leqslant \max(b_1, \cdots, b_n)$, 在冒泡排序的 r 次扫描之后, BOUND 的值是 $\max\{b_i + i \mid b_i \geqslant r\} - r$。

6.[*M22*] 设 $a_1 a_2 \cdots a_n$ 是 $\{1 \cdots n\}$ 的一个排列, 并设 $a_1' \cdots a_n'$ 是它的逆。试证, 对 $a_1 \cdots a_n$ 进行冒泡排序的扫描次数是 $1 + \max(a_1' - 1, a_2' - 2, \cdots, a_n' - n)$。

7.[*M28*] 计算冒泡排序的扫描次数的标准差, 并用 n 和函数 $P(n)$ 来表达它[参见等式(6)和(7)]。

8.[*M24*] 推导等式(8)。

9.[*M48*] 试分析在鸡尾混合排序算法中的扫描次数和比较次数。注: 部分相关信息可参见习题 5.4.8-9。

10.[*M26*] 设 $a_1 a_2 \cdots a_n$ 是 $\{1, 2, \cdots, n\}$ 的一个 2 有序排列。(a)对应的格磁盘通路第 a_i 步的端点坐标是什么(参见图 11)? (b)证明 $a_1 : a_2, a_3 : a_4, \cdots$ 的比较/交换, 对应于按对角线折叠这条通路, 如图 18(b)所示。(c)证明当 $d = 2m - 1$ 时, $a_2 : a_{2+d}, a_4 : a_{4+d}, \cdots$ 的比较/交换, 对应于按对角线之下 m 个单位处的一条直线折叠形成的通路, 如图 18(c)、(d)和(e)所示。

▶11.[*M25*] $\{1, 2, \cdots, 16\}$ 的什么排列使由 Batcher 算法所完成的交换次数取极大值?

12.[24] 假定 MIX 是一台具有操作 AND、SRB 的二进制计算机, 写出算法 M 的一个 MIX 程序。为把表 1 中的 16 个记录排序, 你的程序花费多少时间?

13.[10] Batcher 方法是一个稳定的排序算法吗?

14.[*M21*] 设 $c(N)$ 是通过 Batcher 方法对 N 个元素排序所进行的键码比较次数; 这是步骤 M4 被执行的次数。(a)证明对于 $t \geqslant 1$, 有 $c(2^t) = 2c(2^{t-1}) + (t-1)2^{t-1} + 1$。(b)试求作为 t 的一个函数的 $c(2^t)$ 的简单表达式。提示: 考虑序列 $x_t = c(2^t)/2^t$。

15.[*M38*] 本题的任务是分析习题 4 的函数 $c(N)$, 并求出当 $N = 2^{e_1} + 2^{e_2} + \cdots + 2^{e_r}, e_1 > e_2 > \cdots > e_r \geqslant 0$ 时关于 $c(N)$ 的一个公式。(a)设 $a(N) = c(N+1) - c(N)$。证明 $a(2n) = a(n) + \lfloor \lg(2n) \rfloor$, 以及 $a(2n+1) = a(n) + 1$; 因此

$$a(N)=\binom{e_1+1}{2}-r(e_1-1)+(e_1+e_2+\cdots+e_r)$$

(b)设 $x(n)=a(n)-a(\lfloor n/2\rfloor)$,于是 $a(n)=x(n)+x(\lfloor n/2\rfloor+x(\lfloor n/4\rfloor)+\cdots$。设$y(n)=x(1)+x(2)+\cdots+x(n)$;且设 $z(2n)=y(2n)-a(n)$, $z(2n+1)=y(2n+1)$。证明 $c(N+1)=z(N)+2z(\lfloor N/2\rfloor)+4z(\lfloor N/4\rfloor)+\cdots$。(c)证明 $y(N)=N+(\lfloor N/2\rfloor+1)(e_1-1)-2^{e_1}+2$。(d)现在使 r 保持固定,综合以上所得,并借助指数 e_j 求出关于$c(N)$的一个公式。

16.[*HM42*] 假定 N 是 2 的次幂,求出当应用 Batcher 方法于 N 个不同元素的随机排列时,所需的平均交换次数的渐近值。

▶17.[20] K_0 和 K_{N+1}的值就是在(13)中假设的值,在算法 Q 中何处用了这一事实?

▶18.[20] 说明当所有的输入键码都相等时,算法 Q 中的计算如何进行。如果步骤 Q3 和 Q4 中的“<”号改成“≤”,将发生什么情况?

19.[15] 如果使用一个队列(先进先出),而不是使用栈(后进先出),问算法 Q 是否仍将正确地工作?

20.[*M20*] 作为 M 和 N 的函数,在算法 Q 中最多能有多少元素同时在栈中?

21.[20] 说明当诸键码不同时,为什么算法 Q 的最初的分划阶段要花费(17)中所确定的比较、交换等次数。

22.[*M25*] 令 p_{kN}是当应用算法 Q 于{1,2,…,N}的一个随机排列时,(16)中的量 A 将等于 k 的概率;并令 $A_N(z)=\sum_k pkNz^k$ 是对应的生成函数。试证对于 $N\leqslant M$, $A_N(z)=1$,而对于 $N>M$, $A_N(z)=z(\sum_{1\leqslant s\leqslant N}A_{s-1}(z)A_{N-s}(z))/N$。试求确定其它概率分布 $B_N(z)$、$C_N(z)$、$D_N(z)$、$E_N(z)$、$S_N(z)$的类似递推关系。

23.[*M23*] 令 A_N、B_N、D_N、E_N、S_N 是当对{1,2,…,N}的一个随机排列排序时,(16)中对应量的平均值。试求类似于(18)的这些量的递推关系,并且解这些递推关系以得到(25)。

24.[*M21*] 算法 Q 明显地做了比它所需要的还要多一点的比较,因为我们在步骤 Q3 中能有 $i=j$,而且甚至在 Q4 中有 $i>j$。如果当 $i\geqslant j$ 时,我们避免进行所有的比较,则平均说来,所执行的比较次数 c_N 是多少?

25.[*M20*] 当输入的键码依次是数 1,2,…,N 时,问在程序 Q 的计时中量 A、B、C、D、E 和 S 的精确值是什么(假定 $N>M$)?

▶26.[*M24*] 构造一个输入文件,它使程序 Q 甚至比它在习题 25 中进行得还要慢(试找出一种真正坏的情况)。

27.[*M28*](R. Sedgewick) 考虑算法 Q 的最好情况:寻找{1,2,…,23}的一个排列,当 $N=23$ 和 $M=3$ 时,对它排序所花时间最少。

28.[*M26*] 求类似于(20)的递推关系,它被 Singleton 修改过的算法 Q 的平均比较次数所满足(选择 s 作为{K_1, $K_{\lfloor(N+1)/2\rfloor}$, K_N}的中值,而不是 $s=K_1$)。

29.[*HM40*] 继续习题 28,求 Singleton 的“三值取中”方法中比较次数的渐近值。

30.[25] (P. Shackleton) 当对多个字的键码排序时,许多排序方法随着文件接近它的最后次序而逐渐慢下来,因为相等的和接近相等的键码需要检查几个字才能确定适当的字典编辑次序(见习题 5-5)。通常实践中出现的文件都包含接近相等的键码,所以这个现象对排序时间可能有重大的影响。

试说明怎样推广算法 Q 以避免这个困难;在一个已知对于所有键码其前导的 k 个字都为常数值的子文件中,仅需检查键码的第$(k+1)$个字。

▶31.[20] (C. A. R. Hoare) 假设不是对整个文件排序,而只要确定一个给定的 n 个元素集合

的第 m 个最小者。证明快速排序可修改成实现这一目的，以避免为进行一次完全排序所需要的许多计算。

32.[*M40*] 给出利用习题 31 的"快速查找"方法寻找 n 个元素的第 m 个最小者，所需要进行的平均键码比较次数 C_{nm} 的简单的"封闭形式"的表达式（为简便起见，令 $M=1$；即，不假定使用有关短的子文件的一个特殊技术）。通过 Hoare 的方法求 $2m-1$ 个元素的中间值，所需要的平均比较次数 $C_{(2m-1)m}$ 的渐近特性如何？

▶33.[15] 试设计一个算法，它重新排列一个给定表中的所有数，使得所有负数都在所有非负数之前（这些项不必完全排序，仅需分开负的和非负的）。你的算法应使用最小可能的交换次数。

34.[20] 如何加快基数交换（在步骤 R3 到 R6 中）的位检查循环？

35.[*M23*] 分析频率 A,B,C,G,K,L,R,S 和 X 的值，它们是在利用"情况(i)的输入"的基数交换排序中出现的。

36.[*M27*] 给定数的序列 $\langle a_n\rangle = a_0, a_1, a_2, \cdots$，通过规则

$$\hat{a}_n = \sum_k \binom{n}{k}(-1)^k a_k$$

定义它的二项式转换 $\langle \hat{a}_n\rangle = \hat{a}_0, \hat{a}_1, \hat{a}_2, \cdots$。(a)证明 $\langle \hat{\hat{a}}_n\rangle = \langle a_n\rangle$。(b)求序列 $\langle 1\rangle$；$\langle n\rangle$ 的二项式转换；对于固定的 m，求 $\left\langle \binom{n}{m} \right\rangle$ 的二项式转换；对于固定的 a，求 $\langle a^n\rangle$ 的二项式转换；对于固定的 a 和 m，求 $\left\langle \binom{n}{m} a^n \right\rangle$ 的二项式转换。(c)假设序列 $\langle x_n\rangle$ 满足关系

$$x_n = a_n + 2^{1-n}\sum_{k\geqslant 2}\binom{n}{k}x_k, \quad \text{对于 } n\geqslant 2; \quad x_0 = x_1 = a_0 = a_1 = 0$$

证明

$$x_n = \sum_{k\geqslant 2}\binom{n}{k}(-1)^k \frac{2^{k-1}\hat{a}_k}{2^{k-1}-1} = a_n + \sum_{k\geqslant 2}\binom{n}{k}(-1)^k\frac{\hat{a}_k}{2^{k-1}-1}$$

37.[*M28*] 在习题 36 的意义下，确定所有使得 $\langle \hat{a}_n\rangle = \langle a_n\rangle$ 的序列 $\langle a_n\rangle$。

▶38.[*M30*] 当基数交换应用于"情况(ii)的输入"时，求(29)中诸量的平均值 A_N、B_N、C_N、G_N、K_N、R_N、L_N 和 X_N。试借助 N 和量

$$U_n = \sum_{k\geqslant 2}\binom{n}{k}\frac{(-1)^k}{2^{k-1}-1} \quad V_n = \sum_{k\geqslant 2}\binom{n}{k}\frac{(-1)^k k}{2^{k-1}-1} = n(U_n - U_{n-1})$$

表达你的答案[提示：见习题 36]。

39.[20] (30)中所示的结果指出，当应用基数交换排序于随机输入时，大约花费 $1.44N$ 个分划阶段。证明快速排序决不需要多于 N 个阶段；并说明为什么基数交换往往需要这么多。

40.[21] 说明怎样修改算法 R，使得它对包含许多相等键码的文件排序时，也以相当不错的效率进行工作。

▶41.[30] 试设计对记录 $R_l \cdots R_r$ 进行交换的一个好方法，使得它们被分划成三个块区，且：(i)

对于 $1 \leqslant k < i, K_k \leqslant K$；(ii)对于 $i \leqslant k \leqslant j, K_K = K$；(iii)对于 $j < k \leqslant r, K_k > K$。图解之，最后的安排应是

$<K$	$=K$	$>K$

$l \qquad i \qquad j \qquad r$

42.[*HM32*] 对于任何实数 $c>0$，试证明当对随机数据进行排序时，算法 Q 将做多于 $(c+1)(N+1)H_N$ 次比较的概率小于 e^{-c}（当 c 比如说是 N^{ϵ} 时，这个上限是特别有趣的）。

43.[*HM21*] 证明 $\int_0^1 y^{-1}(e^{-y}-1)dy + \int_1^{\infty} y^{-1}e^{-y}dy = -\gamma$ [提示：考虑 $\lim_{a\to 0+} y^{a-1}$]。

44.[*HM24*] 如正文中所建议的那样，推导(37)。

45.[*HM20*] 说明当 $x>0$ 时，为什么(43)为真。

46.[*HM20*] 给定正整数 s，$0<a<s$，问 $(1/2\pi i)\int_{a-i\infty}^{a+i\infty}\Gamma(z)n^{s-z}dz/(2^{s-z}-1)$ 的值是多少？

47.[*HM21*] 证明 $\sum_{j\geqslant 1}(n/2^j)e^{-n/2^j}$ 是 n 的一个受限函数。

48.[*HM24*] 利用类似于正文中对 U_n 的研究所使用方法，试求在习题 38 中定义的量 V_n 的渐近值，求其诸项直到 $O(1)$ 为止。

49.[*HM24*] 展开 U_n 的渐近公式(47)直到 $O(n^{-1})$。

50.[*HM24*] 当 m 为任何大于 1 的固定数时，求函数

$$U_{mn} = \sum_{k\geqslant 2}\binom{n}{k}(-1)^k\frac{1}{m^{k-1}-1}$$

的渐近值（当 m 为大于 2 的一个整数时，这个量出现于推广基数交换的研究中，以及 6.3 节的检索结构内存的查找算法中）。

▶51.[*HM28*] 证明解决渐近问题的伽玛函数可以用来代替欧拉求和公式，以推导(35)中的量 $r_k(m)$ 的渐近展开（这给了对所有 k 研究 $r_k(m)$ 的一个统一的方法，而无须依赖于诸如正文中介绍的 $g_{-1}(x)=(e^{-x^2}-1)/x$ 的技巧）。

52.[*HM35*] (N.G. de Bruijn) 和数

$$S_n = \sum_{t\geqslant 1}\binom{2n}{n+t}d(t)$$

的渐近特性是什么？其中 $d(t)$ 是 t 的因子的个数（于是，$d(1)=1$，$d(2)=d(3)=2$，$d(4)=3$，$d(5)=2$，等等。这个问题的出现同分析一个树遍历算法有关，见习题 2.3.1-11）。试求 $S_n\Big/\binom{2n}{n}$ 的值，到 $O(n^{-1})$ 的项为止。

53.[*HM42*] 当输入数据由[0..1)中的无穷精度二进数组成，且它们的每位进位都独立地以概率 p 等于 1 时，试分析基数交换所执行的位检查和交换的平均次数（正文中仅仅讨论了 $p=\frac{1}{2}$ 的情况；我们所使用的方法可以推广到任意的 p）。特别是，考虑 $p=1/\phi=.61803\cdots$ 的情况。

54.[*HM24*] (S.O. Rice) 证明 U_n 可被写成

$$U_n = (-1)^n\frac{n!}{2\pi i}\oint_C\frac{dz}{z(z-1)\cdots(z-n)}\frac{1}{2^{z-1}-1}$$

其中 C 是包围点 $2,3,\cdots,n$ 的小的封闭曲线。试把 C 变成以原点为中心的任意大的圆，推导出收敛级数

$$U_n = \frac{(H_{n-1} - 1)n}{\ln 2} - \frac{n}{2} + 2 + \frac{2}{\ln 2}\sum_{m\geqslant 1}\Re(B(n+1, -1+\mathrm{i}bm))$$

其中，$b = 2\pi/\ln 2, B(n+1, -1+\mathrm{i}bm)) = \Gamma(n+1)\Gamma(-1+\mathrm{i}mb)/\Gamma(n+\mathrm{i}bm) = n!/\prod_{k=0}^{n}(k-1+\mathrm{i}bm)$。

▶55.[22] 说明应该如何修改程序 Q，使得分划的元素是 3 个键码(28) 的中值。

56.[*M43*] 当程序已经修改成如习题55中那样取3个元素的中值时，试分析出现于算法Q的运行时间中诸量的平均特性[参考习题 29]。

5.2.3　通过选择进行排序

另一类重要的排序技术是以重复选择的思想为基础的。最简单的选择方法可能如下：

i) 求最小的键码；传送对应的记录到输出区域；然后以值“∞”(假定它高于任何实际的键码)代替这个键码。

ii)重复步骤 i)。这次将选出第二个最小的键码，因为最小的键码已为∞所代替。

iii)继续重复步骤 i)直到已经选择了 N 个记录为止。

一个选择方法要求在排序开始之前所有的输入项目均已出现，而且它顺序地逐个产生最后的输出。这实质上正好与插入法相反，在插入法中，输入被顺序地接收，但在完成排序之前并不知道任何最后的输出。

每次选择出一个新记录时步骤 i)涉及 $N-1$ 次比较，而且它在内存中也需要一个分开的输出区域。但我们显然能做得更好些；通过把被选择的记录同当前占有此记录最后适当位置的那个记录进行交换，我们就可把被选择记录移动到它最后的适当位置，然后在将来的选择中我们就不必再考虑该位置了。因而我们也不需要考虑无穷键码。这个思想就产生了我们的头一个选择排序算法。

算法 S　(直接选择排序)适当重新安排记录 $R_1,\cdots,R_N$；在完成排序后，它们的键码将是有序的 $K_1\leqslant\cdots\leqslant K_N$。排序是以上边指出的方法为基础的，但改为首先选最大元素，紧接着选择第二个最大的，等等，这样做证明是更为方便的。

S1. [对 j 进行循环] 对 $j=N,N-1,\cdots 2$ 执行步骤 S2 和 S3。

S2. [找 $\max(K_1,\cdots,K_j)$] 查一遍 $K_j,K_{j-1},\cdots K_1$，以找出一极大者，设它为 K_i，其中 i 尽可能地大。

S3. [同 R_j 进行交换] 交换记录 $R_i\leftrightarrow R_j$(现在诸记录 $R_j,\cdots,R_N$ 都处于它们的最后位置处)。▌

表 1 示出了本算法应用于 16 个键码示例的情况；在自右至左的查找期间(步骤 S2)，作为极大值被选出的元素以黑体标出。

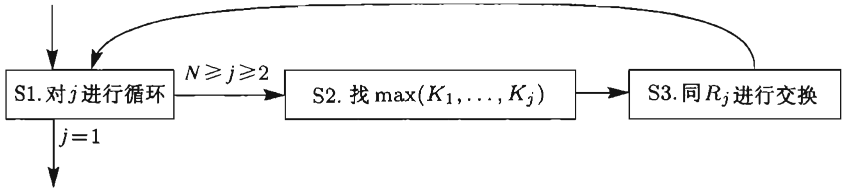


图 21 直接选择排序

表 1 直接选择排序

503	087	512	061	**908**	170	**897**	275	653	426	154	509	612	677	**765**	**703**|
503	087	512	061	703	170	**897**	275	653	426	154	509	612	677	**765**|	908
503	087	512	061	703	170	**765**	275	653	426	154	509	612	**677**|	897	908
503	087	512	061	**703**	170	**677**	275	**653**	426	154	509	**612**|	765	897	908
503	087	512	061	612	170	**677**	275	**653**	426	154	**509**|	703	765	897	908
503	087	512	061	612	170	509	275	**653**|	**426**	**154**|	677	703	765	987	908
…															
061|	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908

对应的 MIX 程序是十分简单的。

程序 S（直接插入排序） 同本章中以前的程序一样，单元 INPUT + 1 到 INPUT + N 中的诸记录按一个全字长键码就地排序；rA≡当前极大值，rI1≡$j-1$，rI2≡k（当前查找的位置），rI3≡i，假定 $N\geq 2$。

```
START  ENT1  N-1          1      S1.对 j 进行循环。 j←N
2H     ENT2  0,1          N-1    S2.找 max(K_1,…,K_j)。 k←j-1
       ENT3  1,1          N-1    i←j
       LDA   INPUT,3      N-1    rA←K_i
8H     CMPA  INPUT,2      A
       JGE   *+3          A      如果 K_i≥K_k，则转换
       ENT3  0,2          B      否则，置 i←k
       LDA   INPUT,3      B      rA←K_i
       DEC2  1            A      k←k-1
       J2P   8B           A      如果 k>0，重复
       LDX   INPUT+1,1    N-1    S3. 同 R_j 进行交换
       STX   INPUT,3      N-1    R_i←R_j
       STA   INPUT+1,1    N-1    R_j←rA
       DEC1  1            N-1
       J1P   2B           N-1    N≥j≥2  ▌
```

这个程序的运行时间依赖于项目数 N；比较的次数 A；以及“自右至左的极大”

的改动次数 B。容易看出，不论输入键码的值为何，有

$$A = \binom{N}{2} = \frac{1}{2}N(N-1) \tag{1}$$

因此仅仅 B 是可变的。尽管直接选择很简单，这个量 B 也是不易精确地分析的。习题 3 到 6 指出

$$B = (\min 0, \text{ave } (N+1)H_N - 2N, \max \lfloor N^2/4 \rfloor) \tag{2}$$

在这种情况下，极大值显得特别有趣。B 的标准差有 $N^{3/4}$ 的阶，见习题 7。

于是程序 S 的平均运行时间是 $2.5N^2 + 3(N+1)H_N + 3.5N - 11$ 个单位，它仅比直接插入(程序 5.2.1S)稍慢些。把算法 S 同冒泡排序(算法 5.2.2B)进行比较是有趣的，因为冒泡排序可以看做是一个选择算法，它有时一次选择出多于一个元素。由于这一原因，冒泡排序通常比直接选择要少做些比较，而且可能显得还优越一些；但事实上，程序 5.2.2B 比程序 S 慢两倍多！冒泡排序逊色的原因是它进行那么多的交换，而直接选择包含非常少的数据移动。

直接选择的精化　有什么途径来改进用于算法 S 中的选择方法呢？例如，步骤 S2 中对于极大值的查找，是否有一个快得多的方法呢？对于后一个问题的回答是否定的！

引理 M　*任何以比较元素对偶为基础的在 n 个元素中寻找极大者的算法，必须至少做 $n-1$ 次比较。*

证明　如果比较少于 $n-1$ 次，则至少有两个元素，它们从未被发现是小于任何其它元素的。因此，就不会知道这两个元素中哪一个是较大的，从而也就不能确定极大者。▌

于是，任何寻找最大元素的选择过程必须至少进行 $n-1$ 步；因此我们可能会怀疑，所有以 n 次重复的选择为基础的排序方法，其步骤的阶数是否都注定为 $\Omega(n^2)$ 次运算？幸而，引理 M 仅适用于头一个选择步骤。随后的选择可以利用前面已获得的信息。例如，习题 8 和 9 指出，对算法 S 稍加改变就把平均次数削减了一半。

考虑表 1 中的 16 个数；节省重复选择时间的一个方法是把它们看做 4 个四元组。我们可以从确定每个组的最大元素开始，这些最大元素的键码是

512，908，653，765

这 4 个元素的最大者是 908，因而也是整个文件的最大者。为了获得第二个最大者，仅仅需要考虑 512，653，715 和包含 908 的组中其它 3 个元素；{170，897，275} 中最大者是 897，而且

512，897，653，765

的最大者是 897。类似地，为了得到第三个最大的元素，先确定{170，275}中的最大者，然后找

512，275，653，765

的最大者。在头一个之后的每一个选择，至多花费 5 个附加的比较。一般说来，如果 N 是一个完全平方，则可以把文件分成为 $\sqrt{N}$ 个 $\sqrt{N}$ 元组；在头一次选择之后，每个选择至多在以前选择了项目的组中花费 $\sqrt{N}-2$ 次比较，加上在“各组尖子”当中的 $\sqrt{N}-1$ 次比较。这一思想被称为二次选择；它的总共执行时间是 $O(N\sqrt{N})$，这比阶 N^2 要好得多。

二次选择首先由 E. H. Friend[*JACM* **3** (1956)，152～154]提出，他指出同一思想可推广到三次、四次选择以及更高次的选择。例如，三次选择把文件划分成 $\sqrt[3]{N}$ 个大组，每一大组包含 $\sqrt[3]{N}$ 个小组，每个小组包含 $\sqrt[3]{N}$ 个记录；执行时间与 $N\sqrt[3]{N}$ 成比例。如果把这个思想引到它最终的结论，则就得到了 Friend 所谓的以一个二叉树结构为基础的“n 次选择”。这个方法的执行时间与 $N\log N$ 成比例，我们称它为树选择。

树选择 借助典型的“淘汰赛”中的对垒，很容易理解树选择排序的原理。例如，考虑图 22 中乒乓球比赛的结果，在底层，Kim 战胜 Sandy，Chris 战胜 Lou；然后在下一轮，Chris 战胜 Kim，等等。

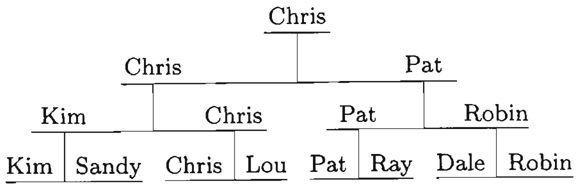


图 22 一场乒乓球锦标赛

图 22 示出 Chris 是 8 个选手中的冠军，而且为确定这一事实要求 $8-1=7$ 次比赛/比较。Pat 不一定是第二个最好的选手。被 Chris 打败的任何一个选手，包括头一轮的失利者 Lou，都可能是第二个最好者，可以通过 Lou 和 Kim 比赛，而这比赛的胜者同 Pat 比，来确定第二个最好的选手；由于我们已经从前几场比赛中记住了这个结构，为找出第二个最好的选手，只需要两场附加的比赛。

一般说来，我们可以在树的根处“输出”选手，然后重新进行比赛，就好像该选手病了而未能发挥出最佳水平。于是原来第二个最好的选手就将升到根的位置；而且在树的上层重新计算胜者。为此目的只有一条通路必须改变。由此得出，为选择第二个最好的选手，只需要少于 $\lceil \lg N\rceil$ 次进一步的比较。同样的过程将找出第三个最好的，等等。因此，如同上面所断言的那样，对于这样一个选择排序，将大体上同 $N\log N$ 成比例。

图 23 示出对于我们的 16 个示例数字进行中的树形选择排序，注意为了知道在哪里插入下一个“$-\infty$”，我们需要知道根处的键码来自何处。因此树的每个分支节点实际上应包含确定相关联的位置的一个指针或下标，而不是键码本身。由此得出，我们需要能存储 N 个输入记录、$N-1$ 个指针和 N 个输出记录(或指向这些记录的指针)的存储空间(当然，如果输出送到磁带或磁盘上，那我们就不需要在高速内存中保留输出记录)。

读者应在此处暂时停下，而来做一做习题 10，因为对于树选择基本原理的一个好的理解，将会更有助于评价我们即将讨论的一些重大改进。

修改树选择的一个方法，实质上是由 K. E. Iverson[*A Programming Lanuage* (Wiley 1962). 223～227]引进的。通过以下列方式的“向前看”就可以去掉对指针的需要，当在树的底层中比赛的优胜者被上移时，在底层处原来优胜者的值就立即以 $-\infty$来代替；而且每当一个优胜者从一个分枝上移到另一个分枝时，它留下的位置即可由最终应当上移到此位置的那个值(即下面两个键码中的较大者)来代替。尽可能重复这个操作就把图 23(a)转换成图 24。

一旦以这种方式建立了树，我们就可以以“由顶向下”的方法，而不是图 23 中由底向上的方法，来进行排序。输出根，然后上移它最大的后裔，然后再上移后者最大的后裔，等等。这个过程开始看起来不大像乒乓球的淘汰赛，而更像是团体选拔系统。

读者应能看出，这个由顶向下的方法有一个优点，即可以避免 $-\infty$ 和 $-\infty$ 的多余比较(由底向上的方法在排序的稍后阶段将发现到处都是 $-\infty$，而由顶向下的方法则在每个阶段中存进一个 $-\infty$后便停止修改树)。

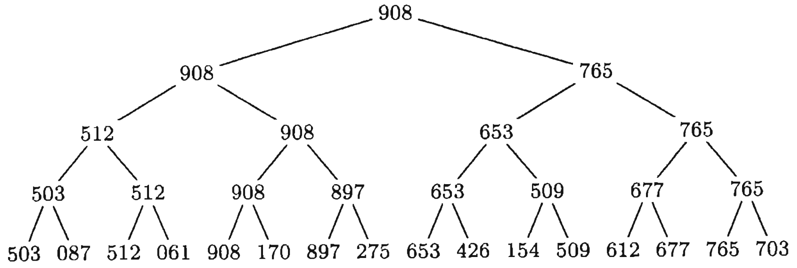
图 23 和图 24 是具有 16 个终端结点的**完备二叉树**(参见 2.3.4.5 小节)，而且以图 25 所示的连续单元来表示这样一株树是方便的。注意，号码为 k 的节点的父亲是节点$\lfloor k/2 \rfloor$，而它的儿子是节点 $2k$ 和 $2k+1$。这就导出了由顶向下方法的另一个优点，因为由顶向下从节点 k 到节点 $2k$ 和 $2k+1$，比起由底向上从节点 k 到节点 $k\oplus 1$ 和$\lfloor k/2 \rfloor$(这里依据 k 是偶数或奇数，$k\oplus 1$ 分别表示 $k+1$ 或 $k-1$)往往要更简便得多。

至今树选择的例子都或多或少地假定了 N 是 2 的一个乘方；但是实际上 N 可以是任意的，因为很容易对任何的 N，构造具有 N 个终端结点的完备二叉树。

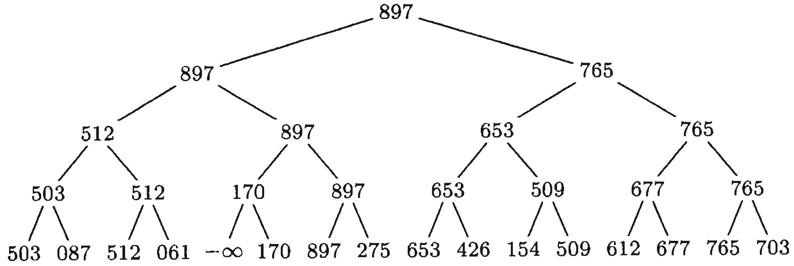
现在遇到了决定性的问题：能否全然不使用 $-\infty$来实施由顶向下的方法？如果图 24 的重复信息全存在完备的二叉树单元 1 到 16 中，而不必包含 $-\infty$的无用“空穴”，岂不是很好吗？稍经思索表明，的确可能达到这一目标，不仅仅是消去 $-\infty$，而且有可能对 N 个记录就地排序，而无须使用任何辅助的输出区域。这是一个重要的排序算法，它的发现者 J. W. J. Williams 为之取名“堆排序”[*CACM* **7** (1964), 347～348]。

堆排序　我们称键码 $K_1, K_2, \cdots, K_N$ 的一个文件为一个“堆”，如果

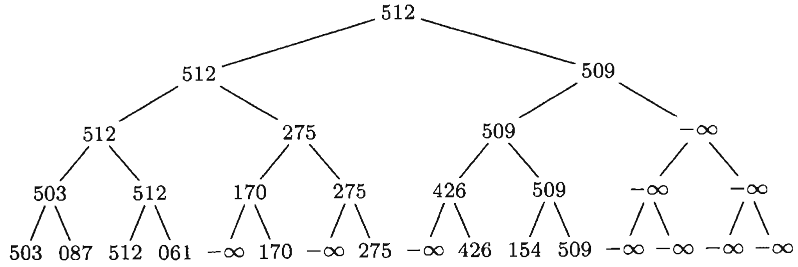
$$K_{\lfloor j/2 \rfloor} \geqslant K_j \quad \text{对于 } 1 \leqslant \lfloor j/2 \rfloor < j \leqslant N \tag{3}$$



(a) 初始配置



(b) 908为−∞所代替，第二高的元素上移至根



(c) 输出了908, 897, 765, 703, 677, 653, 和612之后的配置

图 23 树选择排序的一个例子

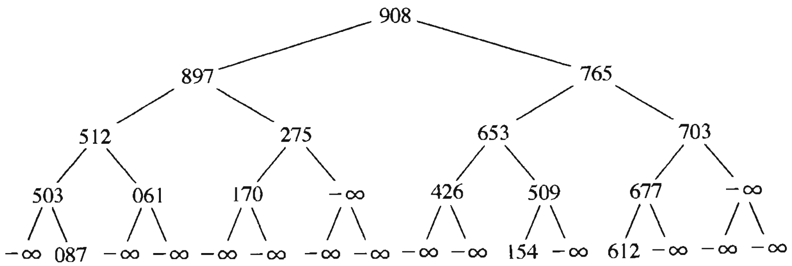


图 24 应用于排序的彼得原理，每个人在这个层次中都上升到他失败的层上

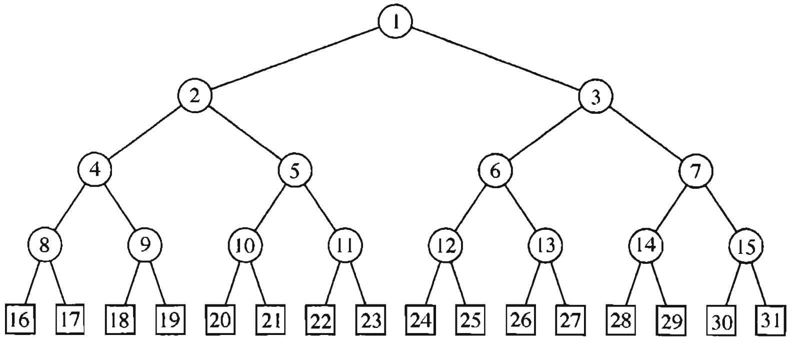


图 25　一株完备二叉树的顺序存储分配

例如 $K_1 \geqslant K_2, K_1 \geqslant K_3, K_2 \geqslant K_4$,等等;这恰是在图 24 中成立的条件,而且,它意味着最大的键码出现在“堆的顶部”

$$K_1 = \max(K_1, K_2, \cdots, K_N) \tag{4}$$

如果能够设法把一个任意的输入文件转换成一个堆,则就可以使用一个如上所述的“由顶向下”的选择过程,来得到一个有效的排序算法。

R. W. Folyd 提出了建立堆的有效方法[*CACM* **7** (1964),701]。假定已经有能力来安排这个文件,使得

$$K_{\lfloor j/2 \rfloor} \geqslant K_1 \quad 对于\ l < \lfloor j/2 \rfloor < j \leqslant N \tag{5}$$

其中 l 是某个$\geqslant 1$ 的数(在原来的文件中,这个条件对于 $l = \lfloor N/2 \rfloor$“空虚地”成立,因为不存在满足条件$\lfloor N/2 \rfloor < \lfloor j/2 \rfloor < j \leqslant N$ 的下标 j)。不难看出应该怎样变换这个文件,使得(5)中的不等式可以推广到$\lfloor j/2 \rfloor = l$ 的情况,使它在以节点 l 为根的整个子树中都成立。因此可以将 l 减 1,直到最终达到条件(3)。Williams 和 Folyd 的这些思想导致了下面的新颖算法,值得仔细地研究。

算法 H(堆排序)　适当地重新安排记录 $R_1, \cdots, R_N$;在完成了排序之后,它们的键码将是有序的,$K_1 \leqslant \cdots \leqslant K_N$。首先,重新安排文件,使得它形成一个堆,然后反复地撤销堆顶,并把它传送到适当的最后位置。假定 $N \geqslant 2$。

H1.[初始化] 置 $l \leftarrow \lfloor N/2 \rfloor + 1, r \leftarrow N$。

H2.[l 或 r 减值] 如果 $l > 1$,则置 $l \leftarrow l - 1, R \leftarrow R_l, K \leftarrow K_l$(如果 $l > 1$,则处于把输入文件变换为一个堆的过程中;如果 $l = 1$,则键码 $K_1 K_2 \cdots K_r$ 已组成一个堆)。否则,置 $R \leftarrow R_r, K \leftarrow K_r, R_r \leftarrow R_1$,以及 $r \leftarrow r - 1$;如果这使得 $r = 1$,则置 $R_1 \leftarrow R$,并且终止这个算法。

H3.[准备“筛选”] 置 $j \leftarrow l$(这时,对于 $r < k \leqslant N$,我们有

$$K_{\lfloor j/2 \rfloor} \geqslant K_j \quad 对于\ l < \lfloor k/2 \rfloor < k \leqslant r \tag{6}$$

而且记录 R_k 处于它最后的位置。步骤 H3～H8 称为“筛选”算法;它们的

效果等价于置 $R_l \leftarrow R$ 然后重新安排 $R_1, \cdots, R_r$，使得条件(6)对于 $l=\lfloor k/2\rfloor$ 也成立。)

H4. [向下进行] 置 $i \leftarrow j$ 和 $j \leftarrow 2j$（在下列步骤中，我们有 $i=\lfloor j/2\rfloor$）。如果 $j<r$，则一直前进到步骤 H5；如果 $j=r$，则转到步骤 H6；如果 $j>r$，则转到 H8。

H5. [找“较大的”儿子] 如果 $K_j < K_{j+1}$，则置 $j \leftarrow j+1$。

H6. [大于 K?] 如果 $K \geqslant K_j$，则转到步骤 H8。

H7. [上移它] 置 $R_i \leftarrow R_j$，并返转到步骤 H4。

H8. [存储 R] 置 $R_i \leftarrow R$（这终止了起始于步骤 H3 的“筛选”算法）。返回到步骤 H2。 ▌

由于 l 和 r 的运动，堆排序有时被描述作“◸”算法。上三角形表示当 $r=N$ 且 l 减小到 1 时堆的建立阶段，而下三角形表示当 $l=1$ 且 r 减小到 1 时的选择阶段。表 2 示出了对 16 个示例数字的堆排序过程（表中的每行示出在步骤 H2 之后的状态，方括号表示位置 l 和 r）。

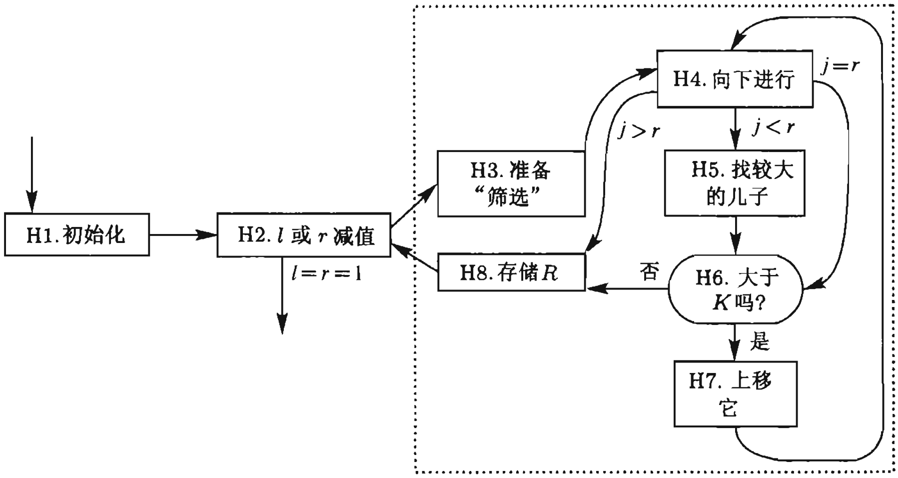


图 26 堆排序，虚线包括的是“筛选”算法

表 2 堆排序示例

K_1	K_2	K_3	K_4	K_5	K_6	K_7	K_8	K_9	K_{10}	K_{11}	K_{12}	K_{13}	K_{14}	K_{15}	K_{16}	l	r
503	087	512	061	908	170	897	275	[653	426	154	509	612	677	765	703]	9	16
503	087	512	061	908	170	897	[703	653	426	154	509	612	677	765	275]	8	16
503	087	512	061	908	170	[897	703	653	426	154	509	612	677	765	275]	7	16
503	087	512	061	908	[612	897	703	653	426	154	509	170	677	765	275]	6	16
503	087	512	061	[908	612	897	703	653	426	154	509	170	677	765	275]	5	16

（续）

K_1	K_2	K_3	K_4	K_5	K_6	K_7	K_8	K_9	K_{10}	K_{11}	K_{12}	K_{13}	K_{14}	K_{15}	K_{16}	l	r
503	087	512	[703	908	612	897	275	653	426	154	509	170	677	765	061]	4	16
503	087	[897	703	908	612	765	275	653	426	154	509	170	677	512	061]	3	16
503	[908	897	703	426	612	765	275	653	087	154	509	170	677	512	061]	2	16
[908	703	897	653	426	612	765	275	503	087	154	509	170	677	512	061]	1	16
[897	703	765	653	426	612	677	275	503	087	154	509	170	061	512]	908	1	15
[765	703	677	653	426	612	512	275	503	087	154	509	170	061]	897	908	1	14
[703	653	677	503	426	612	512	275	061	087	154	509	170]	765	897	908	1	13
[677	653	612	503	426	509	512	275	061	087	154	170]	703	765	897	908	1	12
[653	503	612	275	426	509	512	170	061	087	154]	677	703	765	897	908	1	11
[612	503	512	275	426	509	154	170	061	087]	653	677	703	765	897	908	1	10
[512	503	509	275	426	087	154	170	061]	612	653	677	703	765	897	908	1	9
[509	503	154	275	426	087	061	170]	512	612	653	677	703	765	897	908	1	8
[503	426	154	275	170	087	061]	509	512	612	653	677	703	765	897	908	1	7
[426	275	154	061	170	087]	503	509	512	612	653	677	703	765	897	908	1	6
[275	170	154	061	087]	426	503	509	512	612	653	677	703	765	897	908	1	5
[170	087	154	061]	275	426	503	509	512	612	653	677	703	765	897	908	1	4
[154	087	061]	170	275	426	503	509	512	612	653	677	703	765	897	908	1	3
[087	061]	154	170	275	426	503	509	512	612	653	677	703	765	897	908	1	2

程序 H(堆排序) 通过算法 H,对单元 INPUT + 1 到 INPUT + N 的诸记录进行排序,并对寄存器赋值如下:$rI1 \equiv l-1$,$rI2 \equiv r-1$,$rI3 \equiv i$,$rI4 \equiv j$,$rI5 \equiv r-j$,$rA \equiv K \equiv R$,$rX \equiv R_j$。

```
START  ENT1  N/2          1         H1. 初始化。l←⌊N/2⌋+1
       ENT2  N-1          1         r←N
1H     DEC1  1            ⌊N/2⌋     l←l-1
       LDA   INPUT+1,1    ⌊N/2⌋     R←R_l, K←K_l
3H     ENT4  1,1          P         H3. 准备“筛选”。j←l
       ENT5  0,2          P
       DEC5  0,1          P         rI5←r-j
       JMP   4F           P         转 H4
5H     LDX   INPUT,4      B+A-D     H5. 找“较大的”儿子
       CMPX  INPUT+1,4    B+A-D
       JGE   6F           B+A-D     如果 K_j≥K_{j+1}则转移
       INC4  1            C         否则,置 j←j+1
       DEC5  1            C
```

9H	LDX	INPUT,4	$C+D$	rX←R_j
6H	CMPA	INPUT,4	$B+A$	*H6. 大于 K?*
	JGE	8F	$B+A$	如果 $K \geqslant K_j$,则转到 H8
7H	STX	INPUT,3	B	*H7. 上移它。* $R_i \leftarrow R_j$
4H	ENT3	0,4	$B+P$	*H4. 向下进行。* $i \leftarrow j$
	DEC5	0,4	$B+P$	rI5←rI5 − j
	INC4	0,4	$B+P$	$j \leftarrow j+j$
	J5P	5B	$B+P$	如果 $j<r$,则转到 H5
	J5Z	9B	$P-A+D$	如果 $j=r$,则转到 H6
8H	STA	INPUT,3	P	*H8. 存储 R。* $R_i \leftarrow R$
2H	J1P	1B	P	*H2. l 或 r 减值*
	LDA	INPUT + 1,2	$N-1$	如果 $l=1$,则置 $R \leftarrow R_r, K \leftarrow K_r$
	LDX	INPUT + 1	$N-1$	
	STX	INPUT + 1,2	$N-1$	$R_r \leftarrow R_1$
	DEC2	1	$N-1$	$r \leftarrow r-1$
	J2P	3B	$N-1$	如果 $r>1$,则转到 H3
	STA	INPUT + 1	1	$R_1 \leftarrow R$ ▌

尽管这个程序的长度大约仅是程序 S 的两倍,但当 N 很大时,它更为有效。它的运行时间依赖于

$P = N + \lfloor N/2 \rfloor - 2$,筛选扫描的次数;

A　使键码 K 最终固定在堆的一个内部节点上的筛选扫描次数;

B　在筛选期间被提升的键码总数;

C　步骤 H5 中 $j \leftarrow j+1$ 的次数;

D　步骤 H4 中 $j=r$ 的次数。

这些量将在下面进行分析。实际上,它们围绕其平均值的波动相当小

$$A \approx 0.349N, B \approx N\lg N - 1.87N$$
$$C \approx \frac{1}{2}N\lg N - 0.9N, D \approx \ln N \tag{7}$$

例如,当 $N=1000$ 时,对于随机输入的 4 项实验分别给出 $A=371,351,341,340$;$B=8055,8072,8094,8108$;$C=4056,4087,4017,4083$;$D=12,14,8,13$。因此,总运行时间 $7A+14B+4C+20N-2D+15\lfloor N/2 \rfloor-28$,平均说来近似于 $16N\lg N+0.01N$ 个单位。

根据表 2 我们很难相信,堆排序是非常有效的;在把大的键码藏匿在右边之前左移之!当 N 很小时,它确实是不可思议的排序方法;表 2 中 16 个键码的排序时间是 $1068u$,而简单的直接插入法(程序 5.2.1S)仅花费 $514u$。直接选择法(程序 S)花费 $853u$。

对于较大的 N,程序 H 更为有效。由于程序 H、Shell 排序(程序 5.2.1D)和快速排序(程序 5.2.2Q)这 3 个程序都通过键码的比较进行排序,而且较少或不用辅助存诸,当 $N=1000$ 时,在 MIX 上近似的平均运行时间是(MIX 是典型的计算机,但

特殊的机器当然会产生稍微不同的相对数值)

堆排序　　160000u

Shell 排序　130000u

快速排序　　80000u

当 N 变得更大时,堆排序将优于 Shell 排序,但它的渐近运行时间 $16N\lg N\approx 23.08N\ln N$ 将决不超过快速排序的 $11.67N\ln N$。习题 18 中所讨论的对堆排序的修改,将提高在许多计算机上的处理速度,但即使经过改进也仍然赶不上快速排序。

另一方面,快速排序仅仅是平均有效的,它的最坏情况达到 N^2 阶。而堆排序则具有有趣的性质,即它的最坏情况并不比平均情况坏多少,我们总有

$$A\leqslant 1.5N,\quad B\leqslant N\lfloor\lg N\rfloor,\quad C\leqslant N\lfloor\lg N\rfloor \tag{8}$$

所以不论输入数据的分布如何,程序 H 花费的时间单位将不多于 $18N\lfloor\lg N\rfloor+38N$。堆排序是已见的第一个保证具有 $N\lg N$ 阶的排序方法。下面 5.2.4 小节中讨论的合并排序也有这个性质,但它要求较多的存储空间。

最大者先出　在第二章中我们已经看到,线性表通常可以按照对其所实施的插入和删除操作的特性,合理地划分为不同的类型。插入和删除操作使这些线性表增长和缩小,在每次删除都是撤销表中最年轻的项目这一意义下,一个栈有“后进先出”的特性。所谓最年轻的项目是指在所有当前存在的项目中最新插入的项目。而一个简单的队,在每次删除都是撤销剩下的最老的项目这一意义下,则有“先进先出”的特性。在更复杂的情况下,例如 2.2.5 小节的电梯模拟,需要一个“最小者先出”表,其中每次删除都是撤销有最小键码的项目。这样一个表可以称为一个优先队列,因为每个项目的键码反映了它迅速离开这个表的相对能力。选择排序是优先队列的特殊情况,其中我们在做了 N 次插入之后,接着做 N 次删除。

优先队列有广泛的应用。例如,某些数值迭代方案是以重复选择符合某项测试准则的最大(或最小)值的项目为基础的。选中的项目的参数被改变,并根据参数的新值,按新的测试值重新插入到表中。操作系统通常利用优先队列进行作业调度。习题 15、29 和 36 提出了优先队列其它典型的应用。在后面数章中将出现许多其它的例子。

应如何实现优先队列呢? 显然的方法之一是维持一个排好序的表,其中按键码的次序排列诸项目。于是,插入一个新项目实质上就是我们在 5.2.1 小节插入排序的研究中所讨论的同样问题。另一个处理优先队列的甚至更为显然的方法,是以任意次序保持元素表,每当需要删除具有最大(或最小)键码的元素时,就把这个元素找出来。这两个显然的方法,麻烦都在于:当在表中有 N 项时,插入或删除要求 $\Omega(N)$ 步。所以当 N 很大时,它们是非常消耗时间的。

Williams 在关于堆排序的开创性论文中指出,把堆应用于大的优先队列是很理想的,因为我们可以在 $O(\log N)$ 步中向一个堆插入或从一个堆删除元素;而且,堆的所有元素都紧凑地放置在连续的存储单元中。算法 H 的选择阶段是一系列按最大者先出过程的删除步骤:为删除最大的元素 K_1,我们撤销它,并“筛选”出 K_N 进

入 $N-1$ 个元素的新堆中(如果如同在电梯模拟中那样要一个最小者先出算法,则显然可以改变堆的定义,使得在(3)中“≥”变成为“≤”;为了方便起见,在这里只考虑最大者先出的情况)。一般地说,如果要删除最大的项目,然后插入一个新的元素 x,则可以以 $l=1$, $r=N$,和 $K=x$ 来进行筛选过程。如果希望插入一个元素 x,而无须预先删除堆中的元素,则可以使用习题 16 的“由底向上”的过程。

优先队列的一个链接表示 把优先队列表示为链接二叉树的一个有效方法是 1971 年由(Clark A. Crane 发现的 Technical Report SATN-CS-72 259(Computer Science Department, Stanford University 1972]。他的方法要求在每个记录中有两个链接字段和一个小的计数,比起堆来它有下列优点:

i)当优先队列被处理为一个栈时,插入和删除操作花费的时间是固定的,与队列大小无关。

ii)这些记录决不移动,仅仅改变指针。

iii)仅在 $O(\log N)$ 步之内就能容易地把总共有 N 个元素的两个不相交的优先队列合并成一个优先队列。

稍经修改后的 Crane 的开创性方法如图 27 所示,它展示了一个特殊类型的二叉树结构。每个节点包含一个 KEY(键码)字段,一个 DIST(距离)字段,以及两个链接字段 LEFT(左)和 RIGHT(右)。DIST 字段的值总是等于从该节点到空链接 Λ 的最短通路长度。换句话说,它是从该节点到最接近的空子树的距离。如果定义 DIST(Λ)=0 和 KEY(Λ) = $-\infty$,则在这株树中的 KEY 和 DIST 字段满足下列性质:

$$\mathrm{KEY(P)} \geq \mathrm{KEY(LEFT(P))}, \mathrm{KEY(P)} \geq \mathrm{KEY(RIGHT(P))} \tag{9}$$

$$\mathrm{DIST(P)} = 1 + \min(\mathrm{DIST(LEFT(P))}, \mathrm{DIST(RIGHT(P))}) \tag{10}$$

$$\mathrm{DIST(LEFT(P))} \geq \mathrm{DIST(RIGHT(P))} \tag{11}$$

关系(9)类似于堆的条件(3),它确保树根有最大的键码;而关系(10)恰是上面所述的 DIST 字段的定义。关系(11)是有趣的新方法:它意味着到 Λ 的最短通路总可以通过向右移动而得到。我们说,具有这种性质的树是一个*左倾树*,因为它具有如此强烈地倾向于左边的趋势。

由这些定义,显然,DIST(P) = n 意味着在 P 下面至少存在 2^n 个空子树;否则,将有一条更短的通路从 P 通到 Λ。于是,如果在一个左倾树中有 N 个节点,则从根向下往右边走的通路至多包含 $\lfloor \lg(N+1) \rfloor$ 个节点。通过遍历这一通路,就可能把一个新点插入到优先队列中(见习题 33);因此,在最坏情况下仅仅需要 $O(\log N)$ 步。当树是线性的时(所有 RIGHT 链接都是 Λ)出现最好的情况,而当树完全平衡时出现最坏的情况。

为了撤销在根处的节点,我们只需合并它的两株子树。合并两个分别由 P 和 Q 所指向的不相交的左倾树的操作,从概念上说是简单的:如果 KEY(P)≥KEY(Q),则把 P 取作根,把 Q 同 P 的右子树合并;然后修改 DIST(P),必要时互换 LEFT(P)与 RIGHT(P)。这一过程的详细描述是不难想出来的(见习题 33)。

优先队列技术的比较 当节点的数目 N 很小时,最好使用一种直截了当的线

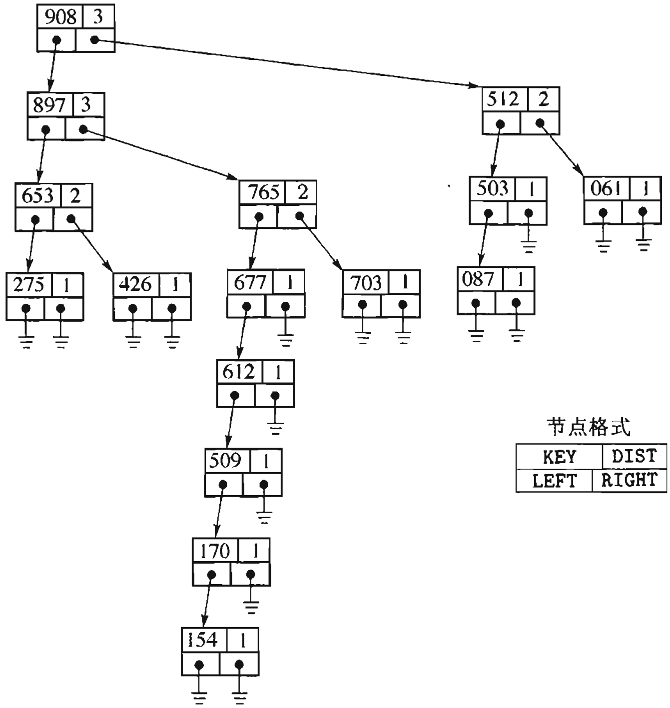


图 27 表示为一株左倾树的优先队列

性表方法，来维持一个优先队列；但当 N 很大时，使用堆或左倾树的 $\log N$ 的方法显然快得多。在 6.2.3 小节，将讨论把线性表表示为平衡树的问题，而这就导致了适合于优先队列的实现的第三个 $\log N$ 方法。因此对这三项技术做一比较是适当的。

已经看出，左倾树操作一般比堆操作稍微快些，但堆花费的存储空间较少，因为堆没有链接字段。平衡树大约花费和左倾树同样多的空间，也许稍微少些；其操作比堆慢些，而且程序设计更复杂，但平衡树结构在某些方面要灵活得多。当使用一个堆或一株左倾树时，我们不易预测，对于具有相等键码的两个项目将发生什么；不可能保证，具有相等键码的项目将以后进先出或先进先出的方式被处理，除非把键码扩展成包括附加的“插入的顺序编号”字段，使得实际上不出现相等的键码。另一方面，通过平衡树，能容易地做出关于相等键码的一致的约定，并且也能做诸如“在 y 之前(或之后)直接插入 x”等操作。平衡树是对称的，以致能在任何时刻删除最大的元素或者最小的元素，而堆和左倾树必然倾向于这边或那边(习题 31 将说明怎样构造对称的堆)。平衡树既可用作查找，也可用作排序；而且，能颇为快速地从一株平衡树撤销连续的元素块区。但是一般说来，为合并两株平衡树需要 $\Omega(N)$步，而左倾树的合并仅需 $O(\log N)$步。

总之，堆使用极小的存储；左倾树对于合并不相交的优先队列是很好的；而如有必要，可以合理的代价利用平衡树的灵活性。

自上文所讨论的 Williams 和 Crane 的开创性工作以来，出现了许多表示优先队列的新方法，除了简单的表、堆、左倾树或平衡树之外，现在程序员有大量的选项可供考虑：

- 分层的树，当所有的键码位于一个给定的范围 $0 \leqslant K < M$ 时，它提供仅需 $O(\log\log M)$ 步的对称优先队列操作（P.van Emde Boas，R.Kaas 和 E.Zijlstra，*Math. Systems Theory* **10**(1977)，99～127]；
- 二项式队列[J.Vuillemin，*CACM* **12**(1978)，309～315；M.R.Brown，*SICOMP* **7**(1978)，298～319]；
- 宝塔式[J.Francon，G.Viennot 和 J.Vuillemin，*FOCS* **19** (1978)，1～7]；
- 成对堆[M.L.Fredman，R.Sedgewick，D.D.Sleator 和 R.E.Tarjan *Algorithmica* **1**(1986)，111～129；J.T.Stako 和 J.S.Vitter *CACM*，**30**(1987)，234～249]；
- 斜堆[D.D.Sleator 和 R.E.Tarjan，*SICOMP* **15**(1986)，52～59]；
- 斐波那契堆[M.L.Fredman 和 R.E.Tarjan，*JACM* **34**(1987)，596～615]以及更一般的 AF-堆[M.L.Fredman 和 D.E.Willard，*J. Computer and system Sci.* **48** (1994)，533～551]；
- 日历队列[R.Brown，*CACM* **31**(1988)，1220～1227；G.A.Davison，*CACM* **32** (1989)，1241～1243]；
- 放松堆[J.R.Driscoll，H.N.Gabow，R.Shrairman 和 R.E.Tarjan，*CACM* **31** (1988)，1343～1354；
- 鱼叉[M.J.Fischer 和 M.S.Patersxon，*JACM* **41**(1994)，3～30]；
- 热队列[B. V. Cherkassky，A. V. Goldberg 和 C. Silverstein，*SODA* **8**(1997)，83～92]；

…

并非所有这些方法都经得起时间的考验；左倾树实际上已经过时，除非应用程序具有很强的面向后进先出特性的趋势。有关二项式队和斐波那契堆的详细实现和解释，可参见*D. E. Knuth. The Stanford GraphBase* (New York：ACM Press，1994).475～489。

***堆排序的分析** 算法 H 是相当复杂的，所以它有可能永远也不会有一个完备的数学分析，但是推导它的若干性质并不会有很大困难。因此，我们将通过稍微详细地研究一个堆的构造来结束本小节。

图 28 所示为具有 26 个元素的一个堆的形状；每个节点都已根据它在此堆中的下标用二进制数标出。图中的星号表示所谓的特殊节点，它位于从 1 到 N 的通路上。

堆最重要的属性之一是它的子树大小的集合。例如，在图 28 中以 1，2，…，26 为根的子树的大小分别是

$$26^*,15,10^*,7,7,6^*,3,3,3,3,$$
$$3,3,2^*,1,1,1,1,1,1,1,1,1,1,1,1,1^* \quad (12)$$

星号表示以特殊节点为根的特殊子树;习题 20 说明,如果 N 的二进表示是

$$N = (b_n b_{n-1} \cdots b_1 b_0)_2, n = \lfloor \lg N \rfloor \quad (13)$$

则特殊子树的大小总是

$$(1b_{n-1}\cdots b_1 b_0)_2, (1b_{n-2}\cdots b_1 b_0)_2, \cdots, (1b_1 b_0)_2, (1b_0)_2, (1)_2 \quad (14)$$

非特殊的子树总是完全平衡的,所以它们的大小总是形如 2^k-1。习题 21 说明非特殊子树的大小恰由

$$\left\lfloor \frac{N-1}{2} \right\rfloor \text{个 } 1, \left\lfloor \frac{N-2}{4} \right\rfloor \text{个 } 3, \left\lfloor \frac{N-4}{8} \right\rfloor \text{个 } 7, \cdots \left\lfloor \frac{N-2^{n-1}}{2^n} \right\rfloor \text{个} (2^n-1) \quad (15)$$

所组成。例如,图 28 含有 12 株大小为 1,6 株大小为 3,2 株大小为 7 和 1 株大小为 15 的非特殊子树。

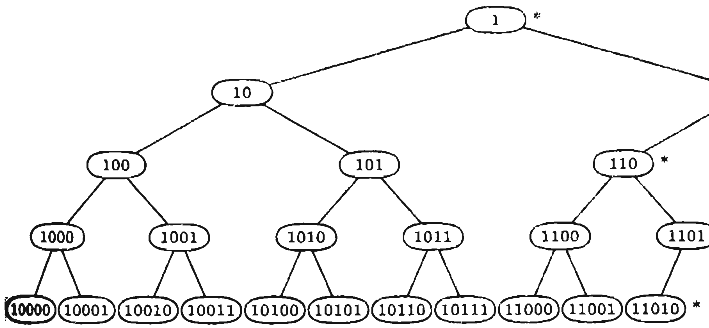


图 28 一个 $26=(11010)_2$ 个元素的堆的图示

设 s_l 是其根为 l 的子树的大小,M_N 是所有这些大小的多重集合 $\{s_1, s_2, \cdots, s_N\}$。利用(14)和(15),对于任何给定的 N,可以容易地计算出 M_N。习题 5.1.4-20 告诉我们,把整数 $\{1,2,\cdots,N\}$ 排列成一个堆的方式总数是

$$N!/s_1 s_2 \cdots s_N = N!/\prod\{s \mid s \in M_N\} \quad (16)$$

例如,放置 26 个字母 $\{A,B,C,\cdots,Z\}$ 到图 28 中,使得竖线保持字母顺序,其方式个数为

$$26!/(26\cdot 10\cdot 6\cdot 2\cdot 1\cdot 1^{12}\cdot 3^6\cdot 7^2\cdot 15^1)$$

我们现在可以来分析算法 H 中的堆建立阶段,即在步骤 H2 中出现条件 $l=1$ 之前的阶段。幸而,可以把对堆建立的研究归结为对独立的筛选操作的研究,这是由下面的定理导出的。

定理 H 如果把算法 H 应用于 $\{1,2,\cdots,N\}$ 的一个随机排列上,则在堆建立阶段,$N!/\Pi\{s \mid s \in M_N\}$ 个可能的堆中,每一个都具有同样的可能性。而且在这个阶

段实施的$\lfloor N/2 \rfloor$个筛选操作的每一个,在下列意义下是一致的,即当达到步骤 H8 时,i 的 s_l 个可能的值中,每一个都是同等可能的。

证明 可以应用数值分析家们也许称之为“向后分析”的方法,来证明定理 H。给定以 l 为根的筛选操作的一个可能结果 $K_1 \cdots K_N$,我们看到,恰有这个文件的 s_l 个以前的配置 $K'_1 \cdots K'_N$,将筛选出此结果。这些以前配置的每一个都有不同的 K'_l 值;因此向后看去,恰有$\{1,2,\cdots,N\}$的 $s_l s_{l+1} \cdots s_N$ 个输入排列,在位置 l 的筛选完成之后,它们产生配置 $K_1 \cdots K_N$。

情况 $l=1$ 是典型的:设 $K_1 \cdots K_N$ 是一个堆,$K'_1 \cdots K'_N$ 是一个文件,当 $l=1$,$K = K'_1$ 时,该文件通过筛选操作变换成 $K_1 \cdots K_N$。如果 $K = K_i$,则必定有 $K'_i = K_{\lfloor i/2 \rfloor}$,$K'_{\lfloor i/2 \rfloor} = K_{\lfloor i/4 \rfloor}$,等等,而对于不在从 1 到 i 的通路上的所有 j,$K'_j = K_j$。反之,对于每个 i,这个构造产生一个文件 $K'_1 \cdots K'_N$,使得:(a)筛选操作把 $K'_1 \cdots K'_N$ 变换成 $K_1 \cdots K_N$;(b)对于 $2 \leqslant \lfloor j/2 \rfloor < j \leqslant N$,$K_{\lfloor j/2 \rfloor} \geqslant K_j$。因此,恰能有 N 个这样的文件 $K'_1 \cdots K'_N$,而且筛选操作是均匀的(这个定理证明的一个例子见习题 22)。 ▌

参照程序 H 的分析中的量 A,B,C,D,可以看到,在大小为 s 的一株子树上的均匀筛选操作,对 A 的平均值的贡献是$\lfloor s/2 \rfloor/s$;对 B 的平均值的贡献是

$$\frac{1}{s}(0+1+1+2+\cdots+\lfloor \lg s \rfloor) = \frac{1}{s}\Big(\sum_{k=1}^{s}\lfloor \lg k \rfloor\Big) = \frac{1}{s}\big((s+1)\lfloor \lg s \rfloor - 2^{\lfloor \lg s \rfloor + 1} + 2\big)$$

(见习题 1.2.4-42);而且根据 s 是偶数还是奇数,它对 D 的平均值的贡献是 $2/s$ 或 0。对 C 的相应的贡献更难确定些,所以把它留给读者(见习题 26)。对于所有筛选操作进行求和,我们求出在堆建立期间 A 的平均值是

$$A'_N = \sum \{\lfloor s/2 \rfloor/s \mid s \in M_N\} \tag{17}$$

对于 B、C 和 D,类似的公式也成立。因此不难精确地计算这些平均值;下表给出了典型的结果:

N	A'_N	B'_N	C'_N	D'_N
99	19.18	68.35	42.95	0.00
100	19.93	69.39	42.71	1.84
999	196.16	734.66	464.53	0.00
1000	196.94	735.80	464.16	1.92
9999	1966.02	7428.18	4695.54	0.00

10000	1966.82	7429.39	4695.06	1.97
10001	1966.45	7430.07	4695.84	0.00
10002	1967.15	7430.97	4695.95	1.73

从渐近的意义上说,可以忽略 M_N 中特殊子树的大小,例如可以求出

$$A'_N = \frac{N}{2}\cdot\frac{0}{1} + \frac{N}{4}\cdot\frac{1}{3} + \frac{N}{8}\cdot\frac{3}{7} + \cdots + O(\log N) = \left(1 - \frac{1}{2}\alpha\right)N + O(\log N) \tag{18}$$

其中

$$\alpha = \sum_{k\geqslant 1}\frac{1}{2^k - 1} = 1.60669\ 51524\ 15291\ 76378\ 33015\ 23190\ 92458\ 04805 - \tag{19}$$

(这个值已由 J.W.Wrench,Jr. 用习题 27 的级数变换计算出来。Paul Erdös 已经证明 α 是一个无理数[*J. Indian Math. Soc.* **12**(1948),63～66],以及 *Peter Borwein* 已经证明许多类似常数的无理性*Proc. Camb. Phil. Soc.* **112**(1992),141～146]。对于很大的 N,可以使用近似公式

$$\begin{aligned} A'_N &\approx 0.1967N + (-1)^N 0.3 \\ B'_N &\approx 0.74403N - 1.3\ln N \\ C'_N &\approx 0.47034N - 0.8\ln N \\ D'_N &\approx (1.8 \pm 0.2)[N\text{ 偶}] \end{aligned} \tag{20}$$

极小值和极大值也容易确定。为建立这个堆,只需要 $O(N)$步(参见习题 23)。

这个理论漂亮地说明了算法 H 的堆建立阶段。但选择阶段是另一回事,它尚有待写出！当对 N 个元素进行堆排序时,A'',B'',C''和 D''表示在选择阶段 A,B,C 和 D 的平均值。对于随机输入,算法 H 的特性受到经验确定的下列平均值,相对小的波动的支配

$$\begin{aligned} A''_N &\approx 0.152N \\ B''_N &\approx N\lg N - 2.61N \\ C''_N &\approx \frac{1}{2}N\lg N - 1.41N \\ D''_N &\approx \lg N \pm 2 \end{aligned} \tag{21}$$

但是对于 D''_N 的特性或对于猜测的常数 0.152、2.61 或 1.41,都还未找到适当的理论解释。然而,B''_N 和 C''_N 的前导项已由 R.Schaffer 和 R.Sedgerwick 以一种非常好的方式确立了;参见习题 30。Schaffer 还证明了 C''_N 可能的极小值和极大值,分别渐近于$\frac{1}{4}N\lg N$ 和$\frac{3}{4}N\lg N$。

习　题

1.[*10*] 直接选择(算法 S)是一个稳定的排序方法吗?

2.[*15*] 为什么在算法 S 中,选择最大的键码,然后次最大的键码,等等;而不是首先找最小的,然后找次小的,等等,证明前者是更为方便的。

3.[*M21*] (a)证明,如果算法 S 的输入是$\{1,2,\cdots,N\}$的一个随机排列,则步骤 S2 和 S3 的第一次迭代产生以 N 结尾的$\{1,2,\cdots,N-1\}$的一个随机排列(换句话说,在 $K_1\cdots K_{N-1}$中$\{1,2,\cdots,N-1\}$的每一个排列的存在都是同等可能的)。(b)因此如果 B_N 表示程序 S 中量 B 的平均值,给定随机排列的输入,则我们有 $B_N=H_N-1+B_{N-1}$[提示:参见等式 1.2.10-(16)]。

▶4.[*M25*] 当 $i=j$ 时算法 S 的步骤 S3 什么也不做;在进行步骤 S3 之前测试是否 $i=j$ 是一个好想法吗? 对于随机输入,在步骤 S3 中条件 $i=j$ 出现的平均次数是多少?

5.[*20*] 当输入是 $N\cdots3\ 2\ 1$ 时,在程序 S 的分析中量 B 的值是什么?

6.[*M29*] (a)设 $a_1a_2\cdots a_N$ 是$\{1,2,\cdots,N\}$的一个排列,它有 C 个循环和 I 个反序,以及当由程序 S 进行排序时有 B 个自右到左的极大值变化。证明 $2B\leqslant I+N-C$[提示:见习题 5.2.2-1]。(b)证明 $I+N-C\leqslant\lfloor N^2/2\rfloor$,因此 B 决不能超过$\lfloor N^2/4\rfloor$。

7.[*M41*] 假定随机输入,试求作为 N 的一个函数的程序 S 中量 B 的方差。

▶8.[*24*] 证明,如果在步骤 S2 中查找 $\max(K_1,\cdots,K_j)$时,按自左到右的次序 $K_1,K_2,\cdots,K_j$ 来考察键码,而不是像在程序 S 中那样按 $K_j,\cdots,K_2,K_1$ 的次序来考察,则通常情况下,有可能减少步骤 S2 的下一次迭代所需要的比较次数。试根据这个想法,写出一个 MIX 程序。

9.[*M25*] 对于随机输入,由习题 8 的算法执行比较的平均次数是多少?

10.[*12*] 在原来的 16 个项目中的 14 项已被输出之后,图 23 中树的配置将是怎样的?

11.[*10*] 在元素 908 被输出之后,图 24 的树的配置将如何?

12.[*M20*] 当使用图 23 的"由底向上"方法把 2^n 个元素的一个文件排成有序时,$-\infty$同$-\infty$将比较多少次?

13.[*20*] (J.W.J.Williams) 算法 H 的步骤 H4 区分 $j<r$,$j=r$ 和$j>r$3 种情况。证明,如果 $K\geqslant K_{r+1}$,则将有可能简化步骤 H4,从而只需做一个两路分支。问应怎样修改步骤 H2,才能确保 $K\geqslant K_{r+1}$的条件贯穿于堆排序过程?

14.[*10*] 证明简单队列是优先队列的特殊情况(说明如何把键码赋予诸元素,使得最大者先出的过程等价于先进先出)。一个栈也是优先队列的一种特殊情况吗?

15.[*M22*] (B.A.Chartres) 设计一个高速算法,它构造一张$\leqslant N$ 的素数的表,并利用一个优先队列来避免除法操作[提示:令优先队列中最小的键码,是比上一个作为素数候选者的奇数要大的最小非素奇数。试使队中元素个数极少]。

16.[*20*] 设计一个有效的算法,它把一个新的键码插入到 n 个元素的一个给定的堆中,产生 $n+1$ 个元素的一个堆。

17.[*20*] 习题 16 的算法可代替算法 H 中"减小 l 到 1"的方法,以用来建立堆。当它们以相同的输入文件开始时,这两个方法是否建立相同的堆?

▶18.[*21*] (R.W.Floyd) 在堆排序的选择阶段,键码 K 势必是十分小的,所以步骤 H6 中几乎所有的比较都发现 $K<K_j$。说明如何修改这个算法,使得在计算的主循环中 K 不同 K_j 进行比较,从而把比较的平均次数削减一半。

19.[*21*] 试设计一个算法,它删去长度为 N 的堆中一个给定的元素,产生长度为 $N-1$ 的一

个堆。

20.[*M20*] 证明(14)给出一个堆中的特殊子树的大小。

21.[*M24*] 证明(15)给出一个堆中的非特殊子树的大小。

▶22.[*20*] 在算法 H 的堆建立阶段,{1,2,3,4,5}的什么排列变换成 5 3 4 1 2?

23.[*M28*] (a)证明在一个筛选算法中扫描的长度 B 决不超过$\lfloor \lg(r/l) \rfloor$。(b)根据(8),在算法 H 的任何具体应用中,B 决不能超过$N\lfloor \lg N \rfloor$。在所有可能的输入文件中,试求 B 作为 N 的一个函数的极大值(你必须证明,存在一个输入文件,使 B 取到这个极大值)。

24.[*M24*] 推导出 B'_N(算法 H 的堆建立阶段的扫描总长度)的标准差的精确公式。

25.[*M20*] 当 $l=1$ 和 $r=N$ 时,如果 $N=2^{n+1}-1$,则在筛选扫描期间对 C 贡献的平均值是多少?

26.[*M30*] 求解习题 25,(a)对于 $N=26$,(b)对于一般的 N。

27.[*M25*] (T.Clausen,1828) 证明

$$\sum_{n\geqslant 1}\frac{x^n}{1-x^n}=\sum_{n\geqslant 1}\frac{1+x^n}{1-x^n}x^{n^2}$$

(置 $x=1/2$ 即得一个用于计算(19)的非常快速地收敛的级数)。

28.[*35*] 以完备的三叉树而不是二叉树为基础,来剖析三叉堆的思想。三叉堆排序比二叉堆更快吗?

29.[*26*] W.S.Brown 设计一个用于多项式或幂级数的乘法$(a_1x^{i_1}+a_2x^{i_2}+\cdots)(b_1x^{j_1}+b_2x^{j_2}+\cdots)$的算法,其中,答案 $c_1x^{i_1+j_1}+\cdots$的系数按照输入系数相乘的次序来生成[提示:使用一个适当的优先队列]。

▶30.[*HM35*] (R.Schaffer 和 R.Sedgewick) 设 h_{nm}是元素$\{1,2,\cdots,n\}$的堆的个数,对于它们来说,堆排序的选择阶段恰好进行 m 次的增进。试证明 $h_{nm}\leqslant 2^m\prod_{k=2}^{n}\lg k$,并且使用这个关系来证明,由算法 H 所实施的增进的平均次数是 $N\lg N+O(N\log\log N)$。

31.[*37*] (J.W.J.Williams) 试证明,如果把两个堆以适当方式"背对背"地放在一起,则有可能维持这样一种结构,其中,在任何时刻都能在 $O(\log N)$步之内删除最小的或者最大的元素(这样一个结构可以叫做一个优先双队)。

32.[*M28*] 如果被排序的键码都不同,试证明,堆排序的增进的个数 B,总是至少为$\frac{1}{2}N\lg N+O(N)$。提示:考虑最大的$\lceil N/2\rceil$个键码的移动。

33.[*21*] 设计一个算法,它把表示为左倾树的两个不相交的优先队列合并成一个。(特别是,如果给定的两队之一只包含单个元素,则你的算法将把它插入到另一个队中)。

34.[*M41*] 如果忽略 KEY 的值,有多少种可能的 N 个结点的左倾树?这个序列以 1,1.2,4,8,17,38,87,203,482,1160,…开始;使用类似于 2.3.4.4-4 的技术,试证明,对于适当的常数 a 和 b,这个数渐近地是 $ab^NN^{-3/2}$。

35.[*26*] 如果 UP 链接被加到一个左倾树上(参见 6.2.3 小节中三重链接树的讨论),则有可能从优先队列内删去任何节点 P 如下:用合并后的 LEFT(P)和 RIGHT(P)代替 P;然后调整 P 诸祖先的 DIST 场,可能会交换左和右子树,直到达到根或者达到一个其 DIST 场未被改变的节点为止。

试证明:如果在这株树中有 N 个节点,即使这株树可以包含非常长的向上通路,这个过程也不会要求改变多于 $O(\log N)$个 DIST 场。

36.[*18*] (最近最少使用的页的替换) 许多操作系统都使用下列类型的算法:对一个节点集合施加两个操作,(i)"使用"一个节点,(ii)用一个新节点替换最近最少使用的节点。试问什么样

的数据结构有利于确认最近最少使用的节点？

37.［*HM32*］令 $e_N(k)$是在有 N 个元素的一个随机堆中，从根到第 k 个最大元素的预期的树式的距离，并令 $e(k)=\lim_{N\to\infty}e_N(k)$。于是 $e(1)=0, e(2)=1, e(3)=1.5$ 和 $e(4)=1.875$。试求 $e(k)$的渐近值到 $O(k^{-1})$的范围内。

38.［*M21*］试求在具有 N 个内部节点的堆或一个完全二叉树内，对于子树大小的多重集合 M_N 的一个简单递推关系。

5.2.4　通过合并进行排序

合并(或者整理)意味着把两个或多个有序文件组成一个单一的有序文件。例如，我们可以合并两个文件 503 703 765 和 087 512 677，得到 087 503 512 677 703 765。实现此合并的一个简单方法是比较两个最小的项目，输出最小的，而后重复同一过程。以

$$\begin{cases}503 & 703 & 765\\ 087 & 512 & 677\end{cases}$$

开始，我们得到

$$087\begin{cases}503 & 703 & 765\\ 512 & 677\end{cases}$$

然后

$$087\quad 503\begin{cases}703 & 765\\ 512 & 677\end{cases}$$

接着

$$087\quad 503\quad 512\begin{cases}703 & 765\\ 677\end{cases}$$

等等。当两个文件之一被取尽时，需要稍注意些。在下列算法中有这一过程的详细描述。

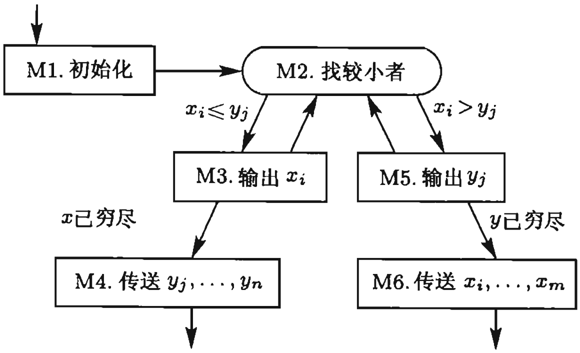


图 29　合并 $x_1\leqslant\cdots\leqslant x_m$ 和 $y_1\leqslant\cdots\leqslant y_n$

算法 M(两路合并)　本算法把有序文件 $x_1\leqslant x_2\leqslant\cdots\leqslant x_m$ 和 $y_1\leqslant y_2\leqslant\cdots\leqslant y_n$ 合并成一个单一文件 $z_1\leqslant z_2\leqslant\cdots\leqslant z_{m+n}$。

M1.[初始化] 置 $i \leftarrow 1, j \leftarrow 1, k \leftarrow 1$。

M2.[找较小者] 如果 $x_i \leqslant y_j$,则转到步骤 M3,否则转到 M5。

M3.[输出 x_i] 置 $z_k \leftarrow x_i, k \leftarrow k+1, i \leftarrow i+1$。如果 $i \leqslant m$,则返回 M2。

M4.[传送 $y_j, \cdots, y_n$] 置 $(z_k, \cdots, z_{m+n}) \leftarrow (y_j, \cdots, y_n)$并终止此算法。

M5.[输出 y_j] 置 $z_k \leftarrow y_j, k \leftarrow k+1, j \leftarrow j+1$。如果 $j \leqslant n$,则返回 M2。

M6.[传送 $x_i, \cdots, x_m$] 置 $(z_k, \cdots, z_{m+n}) \leftarrow (x_i, \cdots, x_m)$并终止此算法。▌

我们将在 5.3.2 小节看到,当 $m \approx n$ 时,这个直截了当的过程,实质上是在一台通常的计算机上进行合并的“最好”方法(而当 m 比 n 小得多时,却有可能设计一个更有效的合并算法,尽管一般说来它们是比较复杂的)。通过在输入文件的末端设置人工的“哨兵”元素 $x_{m+1} = y_{n+1} = \infty$,就可以在不降低很多效率的前提下使算法 M 稍简单些,使其恰恰在输出∞之前停止。关于算法 M 的分析,见习题 2。

算法 M 的总工作量实质上与 $m+n$ 成比例,所以显然,比起排序来,合并是较为简单的问题。而且,我们可以把排序问题归结为合并,因为有可能重复地合并越来越长的子文件直到一切都是有序的为止。可以把这当作排序思想的一个推广:把一个新元素插入到一个有序的文件是合并的特殊情况 $n=1$。如果要加速这个插入过程,则可以考虑一次插入若干个元素,“成批”处理它们,而这自然地导致合并排序的一般思想。从历史上看,合并排序是用于计算机排序早期的方法之一;它早在 1945 年就由约翰·冯·诺依曼提出来了(见 5.5 节)。

我们将在 5.4 节颇为详细地研究合并的问题,并考虑外部排序算法;本节,只专注于在一个高速随机存取存储器内的合并排序问题。

表 1 说明了一种合并排序,它类似于在快速排序、基数交换等方法中使用过的扫描过程的形式,“两头点蜡”:同时从左边和右边考察输入数据,并向中间靠拢。现在暂时忽略这个表的顶上那行,而考虑从行 2 到行 3 的变换。在左边,有递增的路段 503 703 765;在右边向左读,有路段 087 512 677。合并这两个序列得到 087 503 512 677 703 765,它被放置在行 3 的左边。然后行 2 中的键码 061 612 908 同 170 509 897 合并,结果(061 170 509 612 897 908)被记录到行 3 的*右端*。最后,154 275 426 653 同 653 合并——在重叠为害之前就发现它——并把这个结果放到左边,紧跟着前边的路段。本表的行 2 以同样方式由行 1 的原始输入数据形成。

表 1　自然的两路合并排序

503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703
503	703	765	061	612	908	154	275	426	653	897	509	170	677	612	087
087	503	512	677	703	765	154	275	426	653	908	897	612	509	170	061
061	087	170	503	509	512	612	677	703	765	897	908	653	426	275	154
061	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908

表 1 中垂直的线表示路段之间的边界。这些垂线都是所谓的“下坡线”,即它们

前边的元素大于后边的元素。在这个文件的中间,当我们从两个方向读到同一个键码时,一般来说,要出现一个含混的状态;但如果像在下列算法中那样稍加注意,就不致于有问题。由于这个方法利用了它的输入中自然出现的“路段”,故习惯上称为“自然的”合并。

算法 N(自然的两路合并排序) 使用两个存储区域,对记录 $R_1,\cdots,R_N$ 排序,每个存储区域都能容纳 N 个记录。为了方便起见,我们假定,第二个区域的记录是 $R_{N+1},\cdots,R_{2N}$,尽管 R_{N+1}并不必与 R_N 相邻。$R_{N+1},\cdots,R_{2N}$的初始内容是无所谓的。在完成排序之后,键码将是有序的:$K_1\leqslant\cdots\leqslant K_N$。

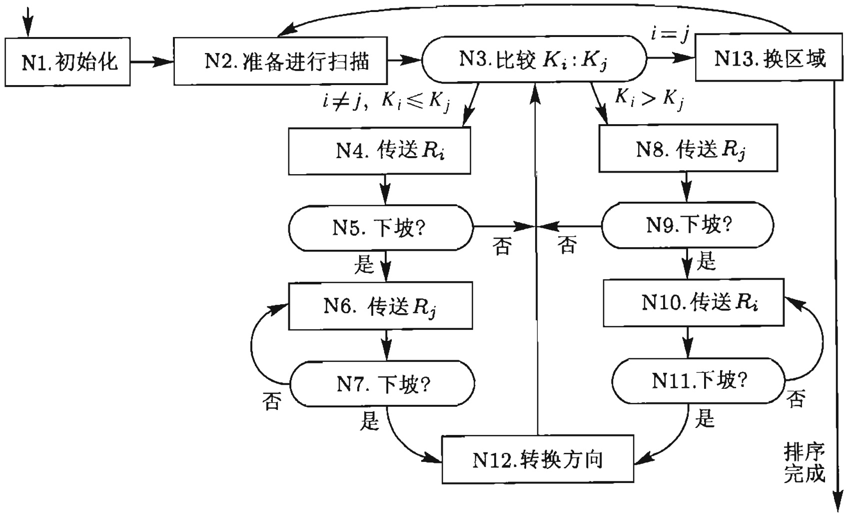


图 30 合并排序

N1.[初始化] 置 $s\leftarrow0$(当 $s=0$ 时,我们把记录从$(R_1,\cdots,R_N)$区域传送到$(R_{N+1},\cdots,R_{2N})$区域;当 $s=1$ 时,以相反方向进行)。

N2.[准备进行扫描] 如果 $s=0$,则置 $i\leftarrow1,j\leftarrow N,k\leftarrow N+1,l\leftarrow2N$;如果 $s=1$,则置 $i\leftarrow N+1,j\leftarrow2N,k\leftarrow1,l\leftarrow N$(变量 i,j,k,l 指向正在读的“源文件”和正在写的“目标文件”的当前位置)。置 $d\leftarrow1,j\leftarrow1$(变量 d 给出输出的当前方向;如果将来还要进行扫描,则置 f 为 0)。

N3.[比较 $K_i:K_j$] 如果 $k_i>K_j$,则转到步骤 N8。如果 $i=j$,则置 $R_k\leftarrow R_i$,并转到 N13。

N4.[传送 R_i](步骤 N4~N7 类似于算法 M 的步骤 M3~M4)置 $R_k\leftarrow R_i,k\leftarrow k+d$。

N5.[下坡?] i 加 1。然后如果 $K_{i-1}\leqslant K_i$,则返回到步骤 N3。

N6.[传送 R_j] 置 $R_k\leftarrow R_j,k\leftarrow k+d$。

N7.[下坡?] j 减 1。然后如果 $K_{j+1}\leqslant K_j$,则返回步骤 N6;否则转到步骤 N12。

N8.［传送 R_j］（步骤 N8～N11 对偶于步骤 N4～N7） 置 $R_k \leftarrow R_j, k \leftarrow k+d$。

N9.［下坡?］j 减 1。然后如果 $K_{j+1} \leqslant K_j$，则返回到步骤 N3。

N10.［传送 R_i］置 $R_k \leftarrow R_i, k \leftarrow k+d$。

N11.［下坡?］i 增加 1。然后如果 $K_{i-1} \leqslant K_i$，则返回到步骤 N10。

N12.［转换方向］置 $f \leftarrow 0, d \leftarrow -d$，并交换 $k \leftrightarrow l$。返回到步骤 N3。

N13.［换区域］如果 $f=0$，则置 $s \leftarrow 1-s$，并返回步骤 N2。否则排序完成；如果 $s=0$，则置 $(R_1, \cdots, R_N) \leftarrow (R_{N+1}, \cdots, R_{2N})$（如果把 $(R_{N+1}, \cdots, R_{2N})$ 作为输出是可接受的，则不必执行后面的拷贝操作）。 ▌

这个算法包含一个巧妙的特性，将在习题 5 中说明它。

虽然编写算法 N 的 MIX 程序并不困难，但是我们无须构造整个程序，就可以推导其特性的基本事实。在随机条件下，输入中的递增路段数大约是 $\frac{1}{2}N$，因为 $K_i > K_{i+1}$ 的概率是 $\frac{1}{2}$；在稍微不同的假设下，关于路段数的详细信息已经在 5.1.3 小节中导出。每次扫描减少一半的路段数（对于像习题 6 那样的不寻常情况除外）。所以扫描数通常将大约是 $\lg \frac{1}{2}N = \lg N - 1$。每次扫描都需要传送全部 N 个记录，而且由习题 2，大部分时间都花费在步骤 N3、N4、N5、N8、N9 上。如果假定键码相等的概率很低，我们可以估算内部循环的时间如下：

步骤		操作	时间
	N3	CMPA,JG,JE	$3.5u$
或者	N4	STA,INC	$3u$
	N5	INC,LDA,CMPA,JGE	$6u$
或	N8	STX,INC	$3u$
	N9	DEC,LDX,CMPX,JGE	$6u$

于是在每次扫描中对于每个记录大约花费 $12.5u$，在平常情况和最坏情况下，总共的运行时间都渐近于 $12.5N \lg N$。这比快速排序的平均时间慢些，而就其花费两倍多的存储空间来说，不应认为它比堆排序好，因为程序 5.2.3H 的渐近运行时间决不会多于 $18N\lg N$。

在算法 N 中，完全通过"下坡"来确定路段之间的边界线。这可能是优点，即：一个主要是递增或主要是递减次序的输入文件可以非常快地被处理。但它减慢了主循环的计算。代替判断下坡，也可人为地确定路段的长度，例如在输入中所有路段的长度都是 1，在第一次扫描之后（最后的路段可能除外）所有路段的长度都是 2，…，在 k 次扫描之后所有路段（可能除开最后的路段）的长度都是 2^k。相对于算法 N

中的“自然”合并而言，称此方法为直接两路合并。

直接两路合并非常类似于算法 N，而且它实质上有同样的流程图；但区别还是不小的，因而最好再次写出整个算法。

算法 S(直接两路合并排序) 如同算法 N 中一样，使用两个存储区域对记录 $R_1,\cdots,R_N$ 排序。

S1. [初始化] 置 $s \leftarrow 0, p \leftarrow 1$(关于变量 s,i,j,k,l,d 的意义见算法 N。这里 p 表示在当前的扫描中有待合并的递增路段的大小；q 和 r 记住在一个路段中未合并的项目数)。

S2. [准备进行扫描] 如果 $s=0$，则置 $i \leftarrow 1, j \leftarrow N, k \leftarrow N, l \leftarrow 2N+1$；如果 $s=1$，则置 $i \leftarrow N+1, j \leftarrow 2N, k \leftarrow 0, l \leftarrow N+1$。然后置 $d \leftarrow 1, q \leftarrow p, r \leftarrow p$。

S3. [比较 $K_i : K_j$] 如果 $K_i > K_j$，则转到步骤 S8。

S4. [传送 R_i] 置 $k \leftarrow k+d, R_k \leftarrow R_i$。

S5. [路段结束?] 置 $i \leftarrow i+1, q \leftarrow q-1$。如果 $q>0$，则转回到步骤 S3。

S6. [传送 R_j] 置 $k \leftarrow k+d$。然后如果 $k=l$，则转到步骤 S13；否则置 $R_k \leftarrow R_j$。

S7. [路段结束?] 置 $j \leftarrow j-1, r=r-1$。如果 $r>0$，则转回到步骤 S6；否则转向 S12。

S8. [传送 R_j] 置 $k \leftarrow k+d, R_k \leftarrow R_j$。

S9. [路段结束?] 置 $j \leftarrow j-1, r \leftarrow r-1$。如果 $r>0$，则转回步骤 S3。

S10. [传送 R_i] 置 $k \leftarrow k+d$。然后如果 $k=l$，则转到步骤 S13；否则置 $R_k \leftarrow R_i$。

S11. [路段结束?] 置 $i \leftarrow i+1, q \leftarrow q-1$。如果 $q>0$，则转回到步骤 S10。

S12. [转换方向] 置 $q \leftarrow p, r \leftarrow p, d \leftarrow -d$，并交换 $k \leftrightarrow l$。如果 $j-i<p$，则返回步骤 S10；否则返回到步骤 S3。

S13. [换区域] 置 $p \leftarrow p+p$。如果 $p<N$，则置 $s \leftarrow 1-s$，并返回到 S2。否则排序完成；如果 $s=0$，则置

$$(R_1,\cdots,R_N) \leftarrow (R_{N+1},\cdots,R_{2N})$$

(当且仅当$\lceil \lg N \rceil$是奇数或 $N=1$ 的平凡情况时，才执行后面的拷贝操作，而不管输入的分布如何。因此，有可能事先预测排序输出的位置，从而避免拷贝)。 ▌

作为本算法的下例，见表 2。稍微使人惊奇的是，当 N 不是 2 的一个乘方时，这个方法工作得很好；正在合并的路段不全都是长度为 2^k 的，对于这些例外情况却还无明文规定(见习题 8)！以前的下坡判断已经被 q 或 r 的减值运算、以及结果是否为 0 的判断所代替；这把 MIX 的渐近运行时间减少到 $11N \lg N$ 单位，比起我们由算法 N 所

表 2　直接两路合并排序

503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703
503	703	512	677	509	908	426	897	653	275	170	154	612	061	765	087
087	503	703	765	154	170	509	908	897	653	426	275	677	612	512	061
061	087	503	512	612	677	703	765	908	897	653	509	426	275	170	154
061	087	154	179	275	426	503	509	512	612	653	677	703	765	897	908

能达到的稍微快些。

实际上,把算法 S 同直接插入结合起来是值得的;我们可以利用直接插入,代替算法 S 的前 4 趟扫描,来对 16 个项目的小组排序,从而避免短的合并造成的相当浪费的簿记操作。如在快速排序情况下所看到的,这种多个方法的结合并不影响渐近的运行时间,但它仍然不失为相当大的改进。

现在从数据结构的观点来研究算法 N 和 S。为什么需要 $2N$ 个记录单元而不是 N 个呢？原因比较简单:我们处理的是可变大小的 4 个表(对于每次扫描要两个“源”表和两个“目标”表);而对于每一对顺序地分配的表,我们使用的是 2.2.2 小节中所讨论的标准的“一起生长”的思想。但是存储空间的一半总是没有用上,稍做考虑就可看出,实际上应该对这 4 个表都使用链接分配。如果对 N 个记录的每一个都加上一个链接字段,利用简单的链接操作而全然不必移动记录,就可以做到合并算法所需要的每一件事！加 N 个链接字段,一般来说比加另外 N 个记录所需要的空间要好,而且减少记录的移动也节约时间。因此,我们应当考虑下述的合并算法。

算法 L(表合并排序)　假定记录 $R_1,\cdots,R_N$ 包含键码 $K_1,\cdots,K_N$,而且链接字段 $L_1,\cdots,L_N$ 能容纳数 $-(N+1)$ 到 $(N+1)$。在文件的开始和结尾的人为记录 R_0 和 R_{N+1} 中,有两个辅助的链接字段 L_0 和 L_{N+1}。本算法是一个“表排序”,它设置链接字段,使得诸记录以递增次序被链接在一起。在排序完成之后,L_0 是具有最小键码记录的下标;而且对于 $1\leqslant k\leqslant N$,L_k 是 R_k 后面的记录下标,如果 R_k 是具有最大键码的记录,则 $L_k=0$(见等式 5.2.1-(13))。

在本算法的过程中,R_0 和 R_{n+1} 作为其子表正被合并的两个线性表的“表头”。一个负的链接表示被排好序的一个子表的结尾;一个零链接表示整个表的结尾。假定 $N\geqslant 2$。

式“$|L_s|\leftarrow p$”指的是“把 L_s 置成 p 或 $-p$,但要保留 L_s 以前的符号”。这个操作很适合于 MIX,但可惜并不适合于大多数计算机;我们可以直截了当地修改这个算法,以得到一个对于大多数机器都同样有效的方法。

L1. [准备两个表] 置 $L_0\leftarrow 1$,$L_{N+1}\leftarrow 2$,对于 $1\leqslant i\leqslant N-2$ 置 $L_i\leftarrow -(i+2)$,并置 $L_{N-1}\leftarrow L_N\leftarrow 0$(我们已经分别建立了包含 $R_1,R_3,R_5,\cdots$ 及 $R_2,R_4,$

$R_6 \cdots$的两个表;负的链接指出每个有序的"子表"仅由一个元素组成。利用在初始数据中可能出现的次序,可以用另一种方法执行这一步,见习题 12)。

L2.［开始新的扫描］置 $s \leftarrow 0, t \leftarrow N+1, p \leftarrow L_s, q \leftarrow L_t$。如果 $q=0$,则本算法终止(在每次扫描期间,p 和 q 都遍历正被合并的表;s 通常指向当前子表的最近被处理的记录,而 t 指向上一次输出的子表的结尾)。

L3.［比较 $K_p : K_q$］如果 $K_p > K_q$,则转 L6。

L4.［推进 p］置 $|L_s| \leftarrow p, s \leftarrow p, p \leftarrow L_p$。如果 $p>0$,则返回 L3。

L5.［完成子表］置 $L_s \leftarrow q, s \leftarrow t$。然后置 $t \leftarrow q$ 且 $q \leftarrow L_q$,一次或多次,直到 $q \leqslant 0$ 为止。最后转向 L8。

L6.［推进 q］(步骤 L6 和 L7 对偶于 L4 和 L5)置 $|L_s| \leftarrow q, s \leftarrow q, q \leftarrow L_q$。如果 $q>0$,则返回 L3。

L7.［完成子表］置 $L_s \leftarrow p, s \leftarrow t$。然后置 $t \leftarrow p$ 和 $p \leftarrow L_p$,一次或多次,直到 $p \leqslant 0$ 为止。

L8.［扫描结束?］(这时,$p \leqslant 0$ 和 $q \leqslant 0$,因为两个指针都分别移动到它们子表的结尾)置 $p \leftarrow -p, q \leftarrow -q$。如果 $q=0$,则置 $|L_s| \leftarrow p, |L_t| \leftarrow 0$ 并返回步骤 L2。否则返回 L_3。 ▌

执行这个算法的一个例子见表 3,其中我们可以看到每次遇到步骤 L2 时的链接情况。使用习题 5.2-12 的方法,即可在这个算法的结尾处重新安排记录 $R_1, \cdots, R_N$,以使它们的键码有序。在表合并和稀疏多项式的加法(见算法 2.2.4A)之间,有着有趣的类似性。

表 3 表合并排序

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
K_j	—	503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703	—
L_j	1	−3	−4	−5	−6	−7	−8	−8	−10	−11	−12	−13	−14	−15	−16	0	0	2
L_j	2	−6	1	−8	3	−10	5	−11	7	−13	9	12	−16	14	0	0	15	4
L_j	4	3	1	−11	2	−13	8	5	7	0	12	10	9	14	16	0	15	6
L_j	4	3	6	7	2	0	8	5	1	14	12	10	13	9	16	0	15	11
L_j	4	12	11	13	2	0	8	5	10	14	1	6	3	9	16	7	15	0

现在来构造算法 L 的一个 MIX 程序,从速度的观点以及空间的观点看,表操作是否有利。

程序 L(表合并排序) 为方便起见,假定记录都是一个字长的,且 L_j 在单元 INPUT + j 的(0:2)字段中,K_j 在(3:5)字段中;rI1≡p, rI2≡q, rI3≡s, rI4≡t, rA≡K_q;$N \geqslant 2$。

```
         L      EQU       0:2                            字段名的定义
ABSL            EQU       1:2
KEY             EQU       3:5
START           ENT1      N-2                1           L1. 准备两个表
                ENNA      2,1                N-2
                STA       INPUT,1(L)         N-2         L_i←-(i+2)
                DEC1      1                  N-2
                J1P       *-3                N-2         N-2≥i>0
ENTA            1                            1
                STA       INPUT(L)           1           L_0←1
ENTA            2                            1
                STA       INPUT+N+1(L)       1           L_{N+1}←2
                STZ       INPUT+N-1(L)       1           L_{N-1}←0
                STZ       INPUT+N(L)         1           L_N←0
                JMP       L2                 1           转 L2
L3Q             LDA       INPUT,2            C''+B'      L3. 比较 K_p:K_q
L3P             CMPA      INPUT,1(KEY)       C
                JL        L6                 C           如果 K_q<K_q 则转 L6
L4              ST1       INPUT,3(ABSL)      C'          L4. 推进 p。|L_s|←p
                ENT3      0,1                C'          s←p
                LD1       INPUT,1(L)         C'          p←L_p
                J1P       L3P                C'          如果 p>0 则转 L3
L5              ST2       INPUT,3(L)         B'          L5. 完成子表。L_s←q
                ENT3      0,4                B'          s←t
                ENT4      0,2                D'          t←q
                LD2       INPUT,2(L)         D'          q←L_q
                J2P       *-2                D'          如果 q>0 则重复
                JMP       L8                 B'          转 L8
L6              ST2       INPUT,3(ABSL)      C''         L6. 推进 q。|L_s|←q
                ENT3      0,2                C''         s←q
                LD2       INPUT,2(L)         C''         q←L_q
                J2P       L3Q                C''         如果 q>0 则转 L3
L7              ST1       INPUT,3(L)         B''         L7. 完成子表。L_s←p
```

	ENT3	0,4	B''	$s \leftarrow t$
	ENT4	0,1	D''	$t \leftarrow p$
	LD1	INPUT,1(L)	D''	$p \leftarrow L_p$
	J1P	* -2	D''	如果 $p>0$ 则重复
L8	ENN1	0,1	B	L8. 扫描完成? $p \leftarrow -p$
	ENN2	0,2	B	$q \leftarrow -q$
	J2NZ	L3Q	B	如果 $q \neq 0$ 则转 L3
	ST1	INPUT,3(ABSL)	A	$\lvert L_s \rvert \leftarrow p$
	STZ	INPUT,4(ABSL)	A	$\lvert L_t \rvert \leftarrow 0$
L2	ENT3	0	$A+1$	L2. 开始新的扫描。$s \leftarrow 0$
	ENT4	N+1	$A+1$	$t \leftarrow N+1$
	LD1	INPUT(L)	$A+1$	$p \leftarrow L_s$
	LD2	INPUT+N+1(L)	$A+1$	$q \leftarrow L_t$
	J2NZ	L3Q	$A+1$	如果 $q \neq 0$ 则转 L3 ▌

利用在这以前已见过多次的技术(见习题 13 和 14),就可以推导这个程序的运行时间;平均说来,它近似于$(10N\lg N + 4.92N)u$,并具有阶为$\sqrt{N}$的很小的标准差。习题 15 说明,以较长的程序为代价,运行时间还可以少到大约 $9N\lg N$。

因此,在进行内部合并时,链接存储技术显然胜过顺序分配:只需要较小的存储空间,而程序运行要快大约 10%~20%。L.J.Woodrum[*IBM Systems J.*, **8**(1969),189~203]和 A.D.Woodall[*Comp. J.*, **13**(1970),110~111]已经发表了类似的算法。

习　题

1.[*21*] 把算法 M 推广到输入文件 $x_{i1} \leqslant \cdots \leqslant x_{im_i}$ 的一个 k 路合并,其中 $i=1,2,\cdots,k$。

2.[*M24*] 假定在 n 个 y 之中插入 m 个 x 的 $\binom{m+n}{m}$ 种安排,每一种都是同等可能的,试求算法 M 中步骤 M2 被执行次数的平均值和标准差。问这个量的极大值和极小值是多少?

▶3.[*20*](更新) 给定记录 $R_1,\cdots,R_M$ 和 $R'_1,\cdots,R'_N$,它们的键码是不同的和有序的,就是说 $K_1<\cdots<K_M$ 和 $K'_1<\cdots<K'_N$,如果我们要求:当第一个文件的记录 R_i 的键码也出现于第二个文件中时,则抛弃 R_i,试说明如何修改算法 M 以得到一个合并的文件。

4.[*21*] 正文中已经注意到,合并排序可以认为是插入排序的一个推广,试说明合并排序如同图 23 中所指出的那样,也同树选择排序密切相关。

▶5.[*21*] 证明在步骤 N6 或 N10 中,i 决不会等于 j(因此在这些步骤中不必判断是否应转向 N13)。

6.[*22*] 求$\{1,2,\cdots,16\}$的一个排列 $K_1,K_2,\cdots,K_{16}$,使得

$$K_2 > K_3,\ K_4 > K_5,\ K_6 > K_7,\ K_8 > K_9,\ K_{10} < K_{11},\ K_{12} < K_{13},\ K_{14} < K_{15}$$

而且算法 N 仅仅用两次扫描完成对这个文件的排序(因为有 8 个或更多的路段,预期在第一次扫描之后至少有 4 个路段,在第二次扫描之后有 2 个路段,而通常至少经过 3 次扫描才能完成排序。如何能通过仅仅两次扫描来完成)。

7.[*16*] 试给出算法 S 所需要的精确扫描次数的公式(作为 N 的一个函数)。

8.[*22*] 在算法 S 中,假定变量 q 和 r 表示当前正在处理的路段中尚未合并的诸元素的长度;q 和 r 两者开始都等于 p,但路段并不都是这样长的。问这怎么能行得通呢?

9.[*24*] 写出算法 S 的一个 MIX 程序,利用类似于程序 L 中的量 $A, B', B'', C', \cdots$ 确定指令频率。

10.[*25*] (D.A.Bell)说明顺序分配的直接两路合并可以通过至多 $\frac{3}{2}N$ 个存储单元完成,而不是像算法 S 中那样需要 $2N$ 个。

11.[*21*] 算法 L 是一个稳定的排序方法吗?

▶12.[*22*] 利用开始时存在的递增路段,修正算法 L 的步骤 L1,使得两路合并是“自然的”(特别是,如果输入已经有序,则在你的步骤 L1 进行之后,步骤 L2 应立即终止这个算法)。

▶13.[*M34*] 按这一章中其它分析的风格,给出对程序 L 的平均运行时间的分析:解释量 A, $B, B', \cdots$,并且说明怎样计算它们的精确平均值。为对表 3 中的 16 个数排序,程序 L 要用多长时间?

14.[*M24*] 设 N 的二进表示是 $2^{e_1} + 2^{e_2} + \cdots + 2^{e_t}$,其中 $e_1 > e_2 > \cdots > e_t \geqslant 0, t \geqslant 1$。证明由算法 L 执行的键码比较极大次数是 $1 - 2^{e_t} + \sum_{k=1}^{t}(e_k + k - 1)2^{e_k}$。

15.[*20*] 算法 L 的手工模拟揭示,它有时做一些多余的动作;步骤 L4 和 L6 中的赋值 $|L_s| \leftarrow p$, $|L_s| \leftarrow q$ 有一半的次数是不必要的,因为在每次由步骤 L4(或 L6)返回到 L3 时,我们有 $L_s = p$ (或 q)。如何能改进程序 L,来消除这些多余的赋值?

16.[*28*] 设计一个类似算法 L 的表合并算法,但它以三路合并为基础。

17.[*20*] (J.M.Carthy) 设 N 的二进表示如同习题 14 中那样,并假定给了我们 N 个记录,这些记录分布在 t 个有序子文件中,其大小分别为 $2^{e_1}, 2^{e_2}, \cdots, 2^{e_t}$。试说明当附加一个新的第 $(N+1)$ 个记录且 $N \leftarrow N+1$ 时,怎样维持这一事态(得到的算法可以称为“联机”合并排序)。

18.[*40*] (M.A.Kronrod 给定仅含两个路段的 N 个记录的一个文件

$$K_1 \leqslant \cdots \leqslant K_M \quad \text{和} \quad K_{M+1} \leqslant \cdots \leqslant K_N$$

有可能在一个随机存取存储器中用 $O(N)$ 个操作对这个文件排序吗?而且不论 M 和 N 的大小如何,只许使用少量固定的附加存储空间(本节描述的所有合并算法都使用与 N 成比例的额外存储空间)。

19.[*26*] 考虑当 $n=5$ 时,如图 31 中所示的具有 n 个“栈”的铁路转辙器网络(在习题 2.2.1-2～2.2.1-5 中,我们曾考虑过一个栈的网络)。如果有 N 列火车从右边进入,我们发现,在一个栈的情况下,这些车的 $N!$ 个排列中仅有比较少的排列出现在左边。

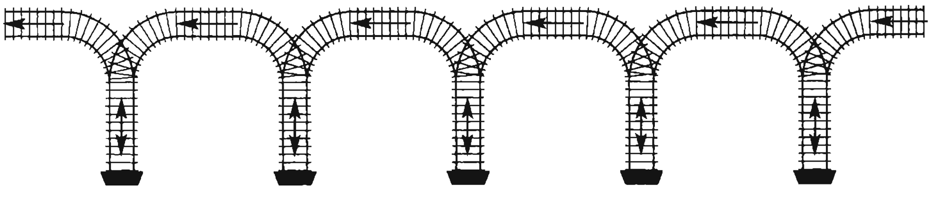


图 31　具有 5 个“栈”的铁路网络

在 n 个栈的网络中，假定有 2^n 列车从右边进入。试证明这些列车的 2^n！种可能排列中的每一种，通过适当的操作序列都可在左边得到。必要时每个栈都比图中所示的要大得多，以便能容纳所有的列车。

20.[*47*] 在习题 2.2.1-4 的符号标记下，通过 n 个栈的铁路网络至多能产生 N 个元素的 a_N^n 个排列；因此为得到所有 N！个排列所需要的栈数至少是 $\log N!/\log a_N \approx \log_4 N$。习题 19 则说明至多需要 $\lceil \lg N \rceil$ 个栈。当 $n \to \infty$ 时，所需要栈数的真实增长率是多少？

21.[*23*](A.J.Smith) 说明怎样扩充算法 L，使得除排序外，它还计算输入排列中出现的反序个数。

22.[*28*](J.K.R.Barnett) 试提出一个方法来加速对于多字键码的合并排序（习题 5.2.2-30 考虑了对于快速排序的类似问题）。

23.[*M30*] 习题 13 和 14 分析了合并排序的“由底向上”或迭代版本，其中对 N 项排序的费用 $c(N)$ 满足下列递推式：

$$c(N) = c(2^k) + c(N - 2^k) + f(2^k, N - 2^k) \quad \text{对于 } 2^k < N \leqslant 2^{k+1}$$

$f(m, n)$ 是把 m 个事物同 n 个事物合并的费用。研究“由顶向下”或分而治之递推式

$$c(N) = c(\lceil N/2 \rceil) + c(\lfloor N/2 \rfloor) + f(\lceil N/2 \rceil, \lfloor N/2 \rfloor) \quad \text{对于 } N > 1$$

此式出现于合并排序以递归形式来进行程序设计时。

5.2.5 通过分布进行排序

我们下面要讨论的一类有趣的排序方法，当以 5.4.7 小节中将讨论的观点来看时，实质上恰是与合并相对。这些方法在电子计算机发现以前，曾被用于对穿孔卡片进行排序许多年。同样的方法也可用于计算机的程序设计，并且由于它是以键码的数字为基础的，因而一般称它为“桶排序”、“基数排序”或“数字排序”。

假设要对 52 张扑克牌进行排序。我们可以定义

$$A < 2 < 3 < 4 < 5 < 6 < 7 < 8 < 9 < 10 < J < Q < K$$

作为面值的一个排序，而对于花色，我们可以定义

$$\clubsuit < \diamondsuit < \heartsuit < \spadesuit$$

如果(i)一张牌的花色小于另一张牌的花色，或(ii)花色相同，但它的面值较另一张牌小（这是对象有序偶之间的一个字典次序的特殊情况，参见习题 5-2），则我们称这张牌是位于另一张牌之前。于是

$$A\clubsuit < 2\clubsuit < \cdots < K\clubsuit < A\diamondsuit < \cdots < Q\spadesuit < K\spadesuit$$

可以用已经讨论过的任何方法来对这些牌进行排序；人们通常使用有些类似于基数交换思想的一种技术：先按它们的花色分成 4 堆，然后拨弄每一堆直到它们变成有序的为止。

但是还有一种更快的方式来处理这叠牌！首先把这些牌仰面分成 13 堆，每种面值一堆。然后通过把牌 A 放在下边，2 朝上放在它们的上面，然后放 3 在上面，等等，最后把 K 面朝上放在上边，来收集这些堆。翻转这些牌使它们面朝下并且再次分堆，这次按 4 种花色分成 4 堆。把这样得到的 4 堆放在一起，黑梅花放在底下，然后方块、红桃和黑桃，这副牌就成为完全有序的了。

同样的思想也可应用于数和字母数据的排序。为什么它有效呢？因为（在扑克牌的例子中）如果两张牌在最后分堆时位于不同的堆中，则它们有不同的花色，所以

具有较低花色的一张在最下面。但是如果两张纸牌有相同的花色(因此它们分在同一堆中),则由于前边的排序它们已处于适当的次序了。换言之,当我们第二次扫描这些纸牌时,对于这 4 堆的每一堆,面值将处于递增的次序。同样的证明可被抽象出来,以指出任何字典次序也可以用这种方式进行排序;关于其细节,见本章开始处习题 5-2 的答案。

刚才描述的排序方法并非一目了然,并且也不清楚是谁最先发现它如此有效。1923 年,IBM 的 Tabulating Machine Company 分部发表的题为"The Inventory Simplified"的 19 页小册子,介绍了在他们的电动排序机上,形成乘积和的一种有趣的数字计划方法:例如,假设我们要用 23～25 列上穿孔的数来乘 1～10 列上穿孔的数,并对大量的卡片把这些乘积加起来。可以首先对第 25 列进行排序,然后使用 Tabulating Machine 求量 $a_1, a_2, \cdots, a_9$,其中 a_k 是对于在第 25 列中有 k 的所有卡片的 1～10 列求和的总数。然后可以对第 24 列进行排序,求类似的总和 $b_1, b_2, \cdots, b_9$;然后对第 23 列,得到 $c_1, c_2, \cdots, c_9$。容易看出所求的乘积和是

$$a_1 + 2a_2 + \cdots + 9a_9 + 10b_1 + 20b_2 + \cdots + 90b_9 + 100c_1 + 200c_2 + \cdots + 900c_9$$

这种穿孔卡片的造表方法自然导致发现首先对最低数字排序的基数排序方法,所以它后来大概已为机器的操作员所熟知。这个排序原理的第一个公开参考文献出现于 L.J.Comrie 关于穿扎卡片设备的早期讨论中[*Transactions of the Office Machinery Users'Assoc.*, *Ltd* (1929),25～37,特别是第 28 页]。

为了处理一台计算机内部的基数排序,我们必须决定如何处理这些堆。假定有 M 个堆,我们可以留出 M 个存储区域,把来自输入区域的每个记录移到它的适当的堆区域中。但这难以令人满意,因为每个区域都必须足够地大以容纳 N 个项目,而这将要求$(M+1)N$ 个记录的空间。因此大多数人都拒绝在计算机内部做基数排序的考虑,直到 H.H.Seward[*Master's thesis*, *M. I. T. Digital Computer Laboratory Report R-232* ,(1954),25～28]指出,可以仅用 $2N$ 个记录的区域和 M 个计数字段,来达到同样的效果,情况才有所改变。只需对这些数据进行初步扫描,以计算 M 堆中的每一堆将有多少个元素;这确切地告诉我们怎样为这些堆分配存储。在"分布计数排序"算法 5.2D 中,已经利用了同样的思想。

于是基数排序可以按如下方式进行:首先按键码的最低位数字(在基数 M 的记法下)进行分布排序,把取自输入区域的记录移到一个辅助区域。然后对于下一个最低位,进行另一次分布排序,把记录移回到原来的输入区域,等等,直到最后(对最高位数字)的扫描把所有的记录排成所希望的次序为止。

如果有一台具有 12 位数字的键码的十进制计算机,并且 N 相当大,则可以选择 $M=1000$(考虑 3 个十进数字作为一个 1000 进制的数字);排序将在 4 次扫描中完成,而不论 N 的大小如何。类似地,如果有一台二进制计算机和一个 40 位的键码,则可以置 $M=1024=2^{10}$,并且在 4 次扫描中完成排序。实际上,每次扫描由三部分组成(计数,分配,移动);E.H.Friend [*JACM* **3**(1956)151]已经建议,在第 k 次扫描中移动记录的同时,为第 $k+1$ 次扫描累加计数,这样,就能以多用 M 个存储单元为代价,把扫描中的两件事情组成在一起。

表 1 说明,对于 $M=10$,这样一个基数排序如何被应用于我们的 16 个示例数

字。一般来说,对于这样小的 N 基数排序没有什么用,举这样一个小例子的目的是要说明这个方法的充分性而不是其效率。

表 1 基数排序

输入区域内容:	503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703
对于个位数字分布的计数:	1	1	2	3	1	2	1	3	1	1						
以这些计数为基础的存储分配:	1	2	4	7	8	10	11	14	15	16						
辅助区域的内容:	170	061	512	612	503	653	703	154	275	765	426	087	897	677	908	509
对于十位数字的分布的计数:	4	2	1	0	0	2	2	3	1	1						
以这些计数为基础的存储分配:	4	6	7	7	7	9	11	14	15	16						
输入区域的内容:	503	703	908	509	512	612	426	653	154	061	765	170	275	677	087	897
对于百位数字的分布的计数:	2	2	1	0	1	3	3	2	1	1						
以这些计数为基础的存储分配:	2	4	5	5	6	9	12	14	15	16						
辅助区域内容:	061	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908

然而,一个敏感的"现代"读者将注意到,对于存储分布进行数字计数的整个思想,都受到顺序数据表示的"旧"思想的束缚;我们知道,链接分配是特别设计用来处理可变大小的多重表的,所以对于基数排序自然应选择链接的数据结构。由于串行地遍历每一堆,因此需要的仅仅是从每个项目到它的后继者的一条单链。而且,决不需要移动这些记录,仅需调整这些链接并轻快地向下通过这个表。需要的存储数量是$(1+\epsilon)N+2\epsilon M$ 个记录,其中 ϵ 是一个链接字段所取的空间数量。这个过程的形式细节是相当有趣的,因为它提供了一个典型数据结构操作的完美例子,它综合了顺序的和链接的分配。

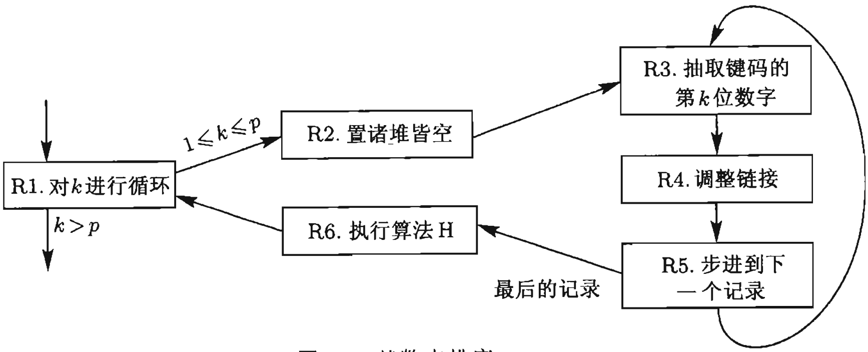


图 32 基数表排序

算法 R(基数表排序) 假定记录 $R_1 \cdots R_N$ 中的每个都包含一个 LINK 字段,它们的键码是 p 元组

$$(a_1, a_2, \cdots, a_p), \quad 0 \leqslant a_i < M \tag{1}$$

其中，次序按字典顺序定义，即

$$(a_1, a_2, \cdots, a_p) < (b_1, b_2, \cdots, b_p) \tag{2}$$

当且仅当对某个 j，$1 \leqslant j \leqslant P$，我们有

$$\text{对所有 } i < j, a_i = b_i \quad \text{但 } a_j < b_j \tag{3}$$

特别是，键码可以想成是以基数 M 的表示法写成的数

$$a_1 M^{p-1} + a_2 M^{p-2} + \cdots + a_{p-1} M + a_p \tag{4}$$

在这种情况下，字典次序相当于非负数的正常次序。键码也可以是字母串，等等。

排序使用 M“堆”记录，以模拟一个卡片排序机动作的方式来进行。诸堆事实上都是第 2 章意义上的队列，因为我们把它们链接在一起，使得它们以先进先出的方式被遍历。对于每个堆，都有两个指针变量 TOP[i]和 BOTM[i]，$0 \leqslant i < M$，而且如同在第 2 章那样，我们假定

$$\text{LINK(LOC(BOTM}[i])) \equiv \text{BOTM}[i] \tag{5}$$

R1. [对 k 进行循环] 开始时，置 $p \leftarrow$ LOC(R_N)，这是指向最后记录的一个指针。然后对于 $k = 1, 2, \cdots, p$ 执行步骤 R2 到 R6（步骤 R2 到 R6 组成一趟“扫描”）。然后这个算法终止，且 p 指向具有最小键码的记录，LINK(P)指向具有次小键码的记录，接着 LINK(LINK(P))，等等；在最后记录中的 LINK 将是 Λ。

R2. [置诸堆皆空] 对于 $0 \leqslant i < M$，置 TOP[i]←LOC(BOTM[i])和 BOTM[i]←Λ。

R3. [抽取键码的第 k 位数字] 令由 P 访问的记录中的键码 KEY(P)是(a_1, a_2, $\cdots$, a_p)；置 $i \leftarrow a_{p+1-k}$，即是这个键码的第 k 个最低位。

R4. [调整链接] 置 LINK(TOP[i])←P，然后置 TOP[i]←P。

R5. [步进到下一个记录] 如果 $k = 1$（第一次扫描）且如果对于某个 $j \neq 1$，P = LOC(R_j)，则置 P←LOC(R_{j-1})并返回到 R3。如果 $k > 1$（随后的扫描），则置 P ←LINK(P)，如果 P≠Λ 则返回到 R3。

R6. [执行算法 H] （现在已把所有的元素都分配到诸堆上）实施以下的算法 H，它把各堆“钩在一起”，形成一个表，以准备进行下次扫描。然后，置 P← BOTM[0]，这是指向已钩起来的表的第一个元素的指针（见习题 3）。 ▌

算法 H（队列的挂钩）给定 M 个队列，它们已各自按照算法 R 的约定链接，本算法至多调整 M 个链接来建立一个单队列，且 BOTM[0]指向第一个元素，并置堆 0 在堆 1 之前，……，在堆 $M-1$ 之前。

H1. [初始化] 置 $i \leftarrow 0$。

H2. [指向堆的顶部] 置 P←TOP[i]。

H3. [下一堆] i 加 1。如果 $i = M$，则置 LINK(P)←Λ，并终止此算法。

H4.［堆为空？］如果 BOTM[i]$=\Lambda$，则回到 H3。

H5.［把诸堆链在一起］置 LINK(P)←BOTM[i]，返回到 H2。▌

图 33 所示为以 $M=10$ 对我们的 16 个例数进行排序时，三次扫描之后的情况，一旦找到处理步骤 R3 和 R5 的逐趟变形的一个适当方法，则算法 R 的 MIX 程序是非常容易编写的。下列程序通过重叠两条指令，在不牺牲内循环任何速度的情况下，做到了这一点。注意 TOP[i]和 BOTM[i]可以组装进同一字中。

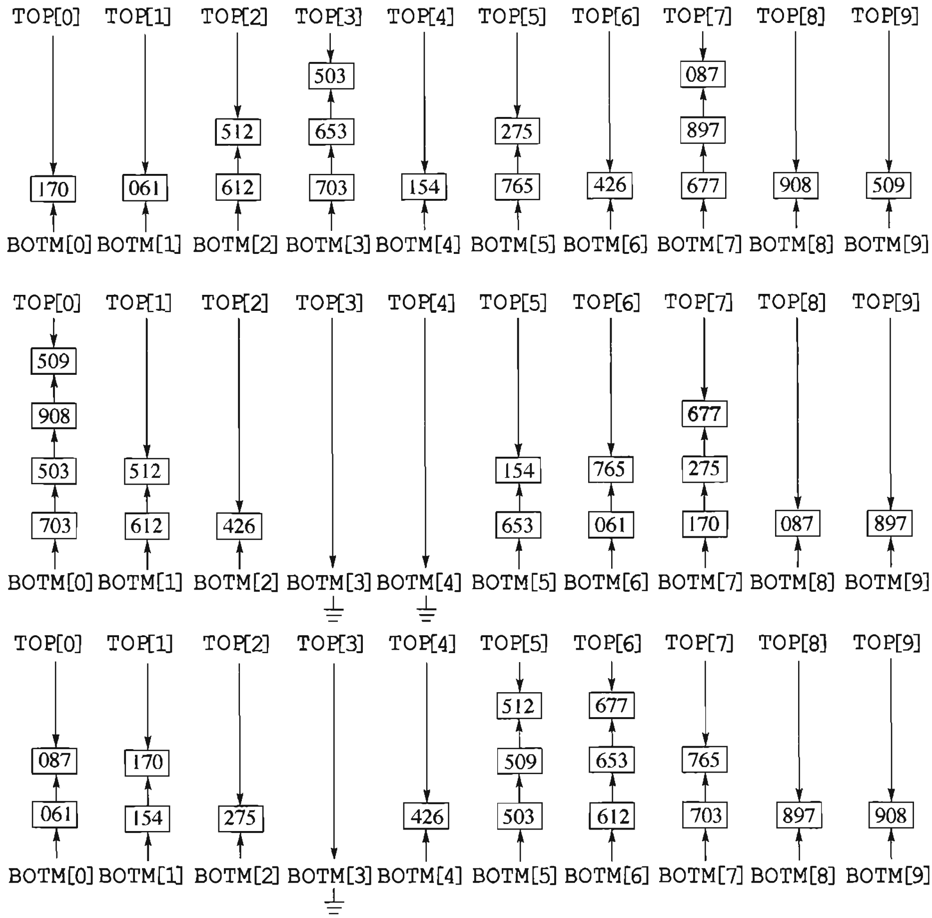


图 33 使用链接分配的基数排序：在每次扫描之后 10 个堆的内容

程序 R（基数表排序） 假定单元 INPUT+1 到 INPUT+N 中的输入有 $p=3$ 个分量(a_1,a_2,a_3)，分别存储在(1:1)，(2:2)和(3:3)字段中(这就假定了 M 是小于或等于 MIX 的字节大小的)。每个记录的(4:5)字段是它的 LINK。对于 $0\leqslant i<M$，令 TOP[i]≡PILES+i(1:2)和 BOTM[i]≡PILES+i(4:5)。使诸链接同单元 INPUT 相关，以便

LOC(BOTM[i] = PILES + i − INPUT 是方便的；为了避免负链接，因此我们要求 PILES 表处于比 INPUT 表高的单元。对变址寄存器赋值如下：rI1≡P，rI2≡i，rI3≡3 − k，rI4≡TOP[i]；在算法 H 期间，rI2≡i − M。

```
LINK   EQU    4:5
TOP    EQU    1:2
START  ENT1   N                   1          R1. 对k进行循环。P←LOC(R_N)
       ENT3   2                   1          k←1
2H     ENT2   M-1                 3          R2. 置诸堆皆空
       ENTA   PILES-INPUT,2       3M         LOC(BOTM[i])
       STA    PILES,2(TOP)        3M         →TOP[i]
       STZ    PILES,2(LINK)       3M         BOTM[i]←Λ
       DEC2   1                   3M
       J2NN   * - 4               3M         M>i≥0
       LDA    R3SW,3              3
       STA    3F                  3          修改第 k 次扫描的指令
       LDA    R5SW,3              3
       STA    5F                  3
3H     [LD2   INPUT,1(3:3)]                  R3. 抽取键码的第 k 位数字
4H     LD4    PILES,2(TOP)        3N         R4. 调整链接
       ST1    INPUT,4(LINK)       3N         LINK(TOP[i])←P
       ST1    PILES,2(TOP)        3N         TOP[i]←P
5H     [DEC1  1]                             R5. 步进到下一个记录
       J1NZ   3B                  3N         如果扫描结束转到 R3
6H     ENN2   M                   3          R6. 执行算法 H
       JMP    7E                  3          以 i←0 转到 H2
R3SW   LD2    INPUT,1(1:1)        N          当 k=3 时,R3 的指令
       LD2    INPUT,1(2:2)        N          当 k=2 时,R3 的指令
       LD2    INPUT,1(3:3)        N          当 k=1 时,R3 的指令
R5SW   LD1    INPUT,1(LINK)       N          当 k=3 时,R5 的指令
       LD1    INPUT,1(LINK)       N          当 k=2 时,R5 的指令
       DEC1   1                   N          当 k=1 时,R5 的指令
9H     LDA    PILES+M,2(LINK)     3M-3       H4. 堆为空?
       JAZ    8F                  3M-3       如果 BOTM[i]=Λ,则转到 H3
       STA    INPUT,1(LINK)       3M-3-E     H5. 把诸堆链在一起
7H     LD1    PILES+M,2(TOP)      3M-E       H2. 指向堆的顶部
8H     INC2   1                   3M         H3. 下一堆。i←i+1
       J2NZ   9B                  3M         如果 i≠M 转到 H4
       STZ    INPUT,1(LINK)       3          LINK(P)←Λ
       LD1    PILES(LINK)         3          P←BOTM[0]
       DEC3   1                   3
       J3NN   2B                  3          对 1≤k≤3 循环 ▌
```

程序 R 的运行时间为 $32N+48M+38-4E$,其中 N 是输入的记录个数,M 是基数(堆数),E 是出现的空堆个数。这非常适合于同以类似假定为基础的其它程序(程序 5.2.1M,5.2.4L)进行对比。这个程序的 p 趟变形将花费 $(11p-1)N+O(pM)$ 个单位时间;计时中的关键因素是内部循环,它包含 5 次对存储的访问和 1 次转移。在一台典型的计算机上,有 $M=b^r$ 和 $p=\lceil t/r \rceil$,其中 t 是在这些键码中基数 b 的数字个数;增加 r 将减少 p,所以这个公式可用来确定 r"最好的"值。

计时中惟一的变量是 E,即在步骤 H4 中发现的空堆个数。如果认为基数为 M 的 M^N 个数字序列中的每一个都是同等可能的,则从 3.3.2D 中对于"扑克检验"的研究知道,每次扫描有 $M-r$ 个空堆的概率是

$$\frac{M(M-1)\cdots(M-r+1)}{M^N}\left\{ {N \atop r} \right\} \tag{6}$$

其中 $\left\{ {N \atop r} \right\}$ 是第二类斯特林数。由习题 5

$$E=\left(\min \max(M-N,0)p, \text{ave} M\left(1-\frac{1}{M}\right)^N p, \max\,(M-1)p\right) \tag{7}$$

"流水线"或"吞嚼"数据的计算机近年来与日俱增。这些机器有多个运算器以及"先行控制"线路,使得存储的访问可以和计算高度重叠,但是它们的效率在出现条件转移指令时显著地下降,除非这个转移几乎总是走同一条路。一个基数排序的内部循环,很适合于这种机器,因为它是典型的"吞嚼"数据形式的一个直接迭代计算。因此,假定 N 不太小而键码不太长的话,比起在这样机器上的任何其它已知的内部排序方法来,基数排序通常要更为有效。

当然,基数排序在键码极其长时不是非常有效的。例如,假定要使用 $M=10^3$,以一个基数排序的 20 次扫描,来对 60 位十进数进行排序;在它们前 9 位数字中,很少有一对数趋向于有相同的键码,所以前 17 次扫描完成的事情很少。在对基数交换排序的分析中,我们发现,当从左边而不是右边考察键码时,没有必要观察键码的许多二进位。因此,我们重新考虑从最高位数字(MSD)而不是从最低位数字(LSD)开始的基数排序的思想。

我们已经说过,首先排最高位数字的基数方法是很自然形成的;事实上,不难看出,为什么邮局即使用这样一种方法来对邮件进行排序。大量的信件可以排序成按不同地理区域分开的邮包,每一个包包含较少量的信件;然后这些信件可独立于其它的包,按越来越细的地理区域继续排序(实际,在它们进一步被排序之前,已被投递到更接近于它们目的地的地方)。这个"分而治之"的原理是十分有吸引力的,而它对穿孔卡片的排序不特别适用的惟一原因,是它毕竟花费太多的时间去跟非常小的堆打交道。算法 R 是相对有效的,尽管它首先考虑最低有效位数字,因为绝不会多于 M 个堆,而且它们仅仅需要钩在一起 p 次。另一方面,利用链接内存,并像算法 5.2.4L 那样以负的链接来表示堆之间的边界,则不难设计首先处理最高有效位的方法(见习题 10)。主要的困难在于空堆趋向于繁衍扩大,并且在最高有效位优先方法中耗费大量时间。

最好的折衷可能是 M.D.MacLaren 所提出来的[*JACM* **13**(1966),404～411],他推荐如算法 R 中那样的首先处理最低位数字的排序,但仅仅应用于最高的那些数字位。这不能完全把文件排好序,但它通常都把文件排成为非常接近有序的,而仅剩很少的反序;因此可用直接插入来完成。对算法 5.2.1M 的分析也可应用于这种情况,所以,如果键码是均匀分布的,在对前 p 个数字进行排序之后,将在文件中平均剩下

$$\frac{1}{4}N(N-1)M^{-p}$$

个反序(见等式 5.2.1-(17)和习题 5.2.1-38)。MacLaren 已经算出每排好一个项目所需的存储访问的平均次数,以及 M 和 p 的最优选择(假定 M 是 2 的乘方,以及这些键码都是均匀分布的,且 $N/M^p \leqslant 0.1$,则对于均匀性的偏离是可以容忍的)是由下表给出的:

N =	100	1000	10000	100000	1000000	10^7	10^8	10^9
最好的 M =	32	128	512	1024	8192	2^{15}	2^{17}	2^{19}
最好的 p =	2	2	2	2	2	2	2	2
$\bar{\beta}(N)$ =	19.3	18.5	18.2	18.1	18.0	18.0	18.0	18.0

此处 $\beta(N)$表示每排好一个项目所需的平均存储访问次数

$$\beta(N) = 5p + 8 + \frac{2pM}{N} + \frac{N-1}{2M^p} - \frac{H_N}{N} \tag{8}$$

当 $N\to\infty$ 时它是有限的,如果取 $p=2$ 和 $M>\sqrt{N}$,则平均的排序时间实际上是 $O(N)$而不是 $N\log N$ 阶的。这个方法是对多重表插入(算法 5.2.1M)的一个改进,后者实质上是 $p=1$ 的情况。对于一个部分地用表排好序的文件的最后重排,习题 12 给出 MacLaren 的一个有趣的方法。

也有可能使用算法 5.2D 和习题 5.2-13 的方法来避免链接字段,以便除了记录本身所需要的空间外,仅仅需要 $O(\sqrt{N})$个存储单元。如果输入记录是均匀分布的,则平均排序时间同 N 成比例。

W.Dobosiewicz 通过使用 MSD 优先的分布排序,直到达到短的子文件为止,同时限制分布过程以确保前 $M/2$ 堆接受 25% 和 75% 之间的记录[参见*Inf. Proc. Letters* **7**(1978),1～6;**8**(1979),170～172];这确保对均匀键码排序的平均时间为 $O(N)$而最坏情况为 $O(N\log N)$。他的论文激励了许多其它的研究者来设计新的地址计算算法,其中最有启发性的可能就是下面由 Markku Tamminen 给出的 2 级方案[*J. Algorithms* **6**(1985),138～144]:假定所有键码都是区间[0..1)中的分数。通过把键码 K 映射到箱$\lfloor KN/8\rfloor$,把 N 个记录分布到$\lfloor N/8\rfloor$个箱子。然后假设箱 k 接受了 N_k 个记录;如果 $N_k\leqslant 16$,通过直接插入来对它排序,否则通过一个类似 MacLaren 的分布 + 插入排序,将其排序到 M^2 个箱中,其中 $M^2\approx 10N_k$。Tamminen 证明了下列值得注意的结果:

定理 T *每当键码为独立随机数,其密度函数 $f(x)$ 有限且对于 $0\leqslant x\leqslant 1$,它们*

是黎曼可积时,则存在一个常数 T,使得刚才描述的排序方法平均说来最多实施 TN 个操作(常数 T 不依赖于 f)。

证明　参见习题 18。直观地说,第一次分布到 $N/8$ 个堆将发现 f 近似为常数的区间;然后第二次分布将使预期的箱大小近似为常数。▌

在由 P.M.McIlroy,K.Bostic 以及 M.D.McIlroy[*Computing Systems* **6**(1993),5～27]所撰写的一篇很有启发的文章中,描述了几个不同版本的基数排序,它们经过很好地调整,适合于对很大的字符串数组进行排序。

习　题

▶1.[*20*] 习题 5.2-13 的算法说明,对于仅有 N 个记录(以及 M 个计数字段)而不是 $2N$ 个记录的区域,怎样来做一个分布排序。这能引出一项对于表 1 中说明的基数排序算法的改进吗?

2.[*13*] 算法 R 是一个稳定的排序方法吗?

3.[*15*] 说明为什么在算法 H 中,虽然堆 0 可以是空的,但 BOTM[0]指向钩起的队中的头一个记录。

▶4.[*23*] 算法 R 把 M 个堆链接在一起成为(先进先出)队,试剖析链接诸堆成为一些栈的思想(图 33 中的箭头将向下进行而不是向上,且 BOTM 表是不必要的)。证明:如果诸堆以适当的次序钩在一起,则有可能得到一个有效的排序方法,这能构成一个更简单或一个更快的算法吗?

5.[*20*] 为使程序 R 能对 8 个字节的键码而不是 3 个字节的键码排序,对它需要做些什么改动? 假定 K_i 的诸最高位字节存储在单元 KEY + i(1:5)中,而 3 个最低位字节就像目前这样存在单元 INPUT + i(1:3)中,在进行了这些改动之后,该程序的运行时间是多少?

6.[*M24*] 设 $g_{MN}(z)=\sum p_{MNk}z^k$,其中 p_{MNk} 是在一个随机基数排序扫描把 N 个元素分成 M 个堆之后,恰出现有 k 个空堆的概率。(a)证明 $g_{M(N+1)}(z)=g_{MN}(z)+((1-z)/M)g'_{MN}(z)$。(b)使用此关系式求出这个概率分布的平均值和方差(作为 M 和 N 的一个函数)的简单表达式。

7.[*20*]　试讨论算法 R 和基数交换排序(算法 5.2.2R)之间的相似性和区别。

▶8.[*20*]　正文中讨论的基数排序算法假定,被排序的所有键码都是非负的,当这些键码是以 2 的补码或 1 的补码符号表示的数时,对诸算法应做什么改动?

9.[*20*] 继续习题 8,当键码是以带正负号的量表达的数时,对诸算法应做些什么改动?

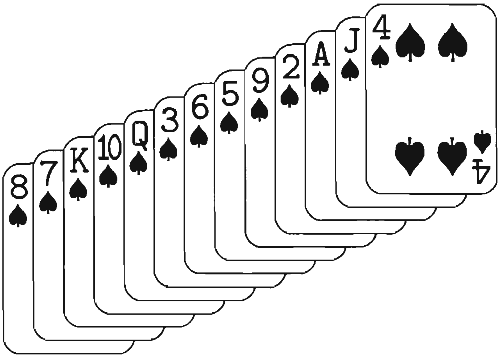
10.[*30*] 设计一个使用链接内存的、有效的首先处理最高位数字的基数排序算法(随着子文件长度的减小,减小 M 并对非常短的子文件使用一个非基数的方法是明智的)。

11.[*16*] 表 1 中所示的 16 个输入数以 41 个反序开始;在排序完成之后,当然已无剩下的反序。如果省略第一次扫描,仅仅对十位和百位数字进行基数排序,则在文件中将存在多少个反序?如果省略第一次扫描和第二次扫描,则将存在多少个反序?

12.[*24*] (M.D.MacLaren) 假设仅将算法 R 应用于实际的键码的前 P 位数字,则当我们以链接的次序来读它时,这个文件已接近于有序了。但是其前 P 位数字相同的那些键码可能仍是无序的。试设计一个算法,它适当地重新安排诸记录,使得它们的键码都成为有序的,$K_1\leqslant K_2\leqslant\cdots\leqslant K_N$[提示:文件被完全排序的特殊情况出现于习题 5.2-12 的答案中;有可能不影响效率而将它与直接插入组合起来,因为在文件中只保留了少量的反序]。

13.[40] 请实现本节结尾处提出的内部排序方法，写一个子程序，它用 $O(N)$ 个时间单位对随机数据排序，且仅使用 $O(\sqrt{N})$ 个附加存储单元。

14.[22] 扑克牌序列



可以在两次扫描中排序成从顶到底递增的次序 A 2…J Q K，且只使用两个堆作为中间存储：把这些牌面朝下分成分别含 A 2 9 3 10 和 4 J 5 6 Q K 7 8（从底到顶）的两堆；然后把第二堆放在第一堆的上面，翻过来使这些牌面朝上，把这些牌分成两堆，A 2 3 4 5 6 7 8，9 10 J Q K。把这两堆组合在一起，让它们面朝上，你就大功告成了。

证明上边的纸牌序列不可能在两次扫描中排序成为从顶到底的递减次序 K Q J…2 A，即使你使用多达三个堆作为中间存储也不行。（处理必须总从纸牌的顶部进行，当它们被处理时，把这些纸牌翻成面朝下。由顶到底相当于图中的自右到左）。

15.[M25] 当所有纸牌必须面朝上而不是面朝下地处理时，考虑习题 14 的问题。于是，可用一次扫描把递增次序转换成递减次序。问需要多少次扫描？

16.[25] 试设计把 m 个字母的字母表上的串 $\alpha_1 \cdots \alpha_n$，排序成为词典次序的一个算法。该算法的总运行时间应当是 $O(m+n+N)$，其中 $N=|\alpha_1|+\cdots+|\alpha_n|$ 是所有串的总长度。

17.[15] 在由 Tamminen 提出的二级分布排序（参见定理 T）中，为什么类似 MacLaren 的方法被用于第二级而不是第一级上？

18.[HM26] 试证明定理 T。提示：首先证明当把 MacLaren 的分布 + 插入算法应用于对于 $0 \leqslant x \leqslant 1$，其概率密度函数满足 $f(x) \leqslant B$ 的独立随机键码时，平均说来，它做 $O(NB)$ 次运算。

For sorting the roots and words
we had the use of 1100 lozenge boxes,
and used trays for the forms.
—GEORGE V. WIGRAM(1843)

5.3 最优排序

既然我们已经分析了内部排序的大量方法，现在应转来考虑一个更广泛的问题：什么是进行排序的最好方法？能否对可能达到的极大排序速度给出极限、使得一个程序员无论如何灵巧也不能超过它呢？

当然，排序并没有什么最好的方式。因为若想达到这一目的，首先就必须精确

地定义什么叫“最好”，但是，又不存在一种最好的方式来定义“最好”。在 4.3.3 小节、4.6.3 小节和 4.6.4 小节中，曾经讨论过算法理论最优性的类似问题，其中考虑了高精度的乘法和多项式的计算。在每种情况下，都有必要阐述“最好的”算法的一种相当简单的定义，以便给出使问题得以求解的充分的结构。而且在每种情况下，都遇到了如此困难的并因此尚未完全解决的有趣问题。对于排序来说情况也是这样，虽已经获得了某些非常有趣的发现，但仍然有许多困难的问题尚未解决。

关于排序固有的复杂性的研究通常针对如下一些问题，即当把 n 个项目排序时，或者把 m 个项目和 n 个项目合并时，或者选择一个未编序的 n 个项目集合的第 t 个最大者时，我们如何把对诸键码的比较次数极小化？5.3.1 小节、5.3.2 小节和 5.3.3 小节泛泛地讨论了这些问题，而 5.3.4 小节则在有趣的限制下讨论了类似的问题，其中所做的限制是：比较的形式必须事先予以基本固定。一些同最优排序有关的其它类型的有趣的理论问题，出现在 5.3.4 小节的习题中，以及在外部排序的讨论中(5.4.4 小节，5.4.8 小节和 5.4.9 小节)。

As soon as an Analytical Engine exists,
it will necessarily guide the future course of the science.
Whenever any result is sought by its aid,
the question will then arise—
By what course of calculation can these
results be arrived at by the machine
in the shortest time?
—CHARLES BABBAGE(1864)

5.3.1 极少比较排序

将 n 个元素排序所需的键码比较要极小次数，显然为 0，因为我们已经知道基数方法不做任何比较。事实上，有可能编写出能进行排序的 MIX 程序，使得它们不包含任何条件转移指令(见本章开始的习题 5～8)！我们也已经了解了若干排序方法，它们实质上是以键码的比较为基础的，然而它们的运行时间实际上取决于其它的因素，如数据移动、簿记操作等。

因此，计算比较次数显然不是测量排序方法有效性的惟一方式。但是，仔细地研究比较次数，无论如何是一桩乐事，因为这个课题的理论研究会给我们带来大量的洞察排序过程本性的有用知识，也会帮助我们增进在其它情况下解决可能碰到的更为普遍的问题的能力。

为了排除完全不做比较的基数排序方法，如同本章开始时所讨论的那样，我们将把讨论局限于仅仅以键码之间的一个抽象线性次序关系“<”为基础的排序技术。为了简便起见也把讨论限制在不同的键码的情况，从而对任何 K_i 和 K_j 的比较，只有两种可能的结果：或者 $K_i > K_j$，或者 $K_i < K_j$(至于把本理论扩充到允许有相等键码的一般情况，见习题 3～12。如果对于基于比较的方法没有限制，对于对整数进

行排序所需要的最坏情况的运行时间的上限，请参见 Fredman 和 Willard，J. Computer and Syst. Sci. **47** (1993)，424～436，Ben-Amram 和 Galil，*J. Comp. Syst. Sci.* **54** (1997)，345～370 和 Thorup，*SODA* **9** (1998)，550～555)。

通过比较进行排序的问题，也可以其它等价的方式来加以表达。如给定 n 个不同的砝码和一台天平，可以提出，当天平的每个盘子仅能容纳一个砝码时，按重量的大小顺序完全排列这些砝码所需要的最少称重次数问题。又或者，给出一项锦标赛中的一组 n 个选手，可以问及，假定选手们的实力可以线性地排序（没有平局）时，足以把所有竞赛者排名次的最少比赛次数的问题。

所有满足上述限制的 n 个元素的排序方法，都可借助如图 34 所示的扩展二叉树结构来表示。每个*内部节点*（画作一个圆圈）包含两个下标“$i:j$”，表示 K_i 同 K_j 的一个比较。这个节点的左子树表示当 $K_i<K_j$ 时所需采取的动作，而右子树则表示当 $K_i>K_j$ 时所要采取的动作。该树的每个*外部节点*（画作一个方框）包含 $\{1,2,\cdots,n\}$ 的一个排列 $a_1a_2\cdots a_n$，表示已经建立起次序

$$K_{a_1}<K_{a_2}<\cdots<K_{a_n}$$

这一事实（如果我们考察从根到这个外部节点的路径，则对于 $1\leqslant i<n$ 的 $n-1$ 个关系 $K_{a_i}<K_{a_{i+1}}$ 中的每一个，都将是这条路径上某个比较 $a_i:a_{i+1}$ 或 $a_{i+1}:a_i$ 的结果）。

因此图 34 表示首先比较 K_1 和 K_2 的一个排序方法；如果 $K_1>K_2$，则它就（通过右子树）去进行 K_2 同 K_3 的比较，接着如果 $K_2<K_3$，则它就比较 K_1 和 K_3；最后，如果 $K_1>K_3$，便知道了 $K_2<K_3<K_1$。一个实际的排序算法通常也将在这个文件中移动键码，但我们在这里仅对比较感兴趣，所以忽略了所有的数据移动。在这株树中，K_i 同 K_j 的比较总是意味着*原来的*键码 K_i 同 K_j，而不是在这些记录已经被重排之后当前可能占有文件的第 i 个和第 j 个位置的键码。

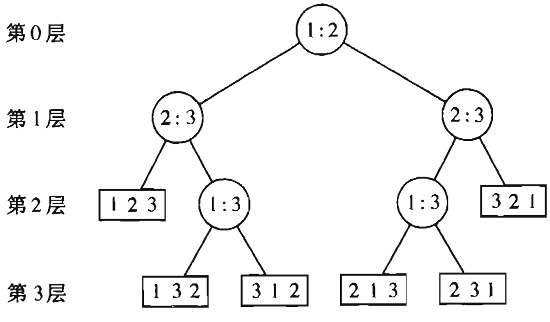


图 34　对 3 个元素排序的一株比较树

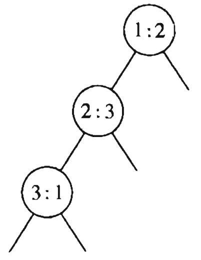


图 35　多余比较的例子

有可能做了多余的比较；例如，在图 35 中，没有理由比较 3:1，因为 $K_1<K_2$ 和 $K_2<K_3$ 意味着 $K_1<K_3$。没有排列能对应于图 35 中节点(3:1)的左子树，所以算法的该部分将永不执行！由于我们的兴趣在于把比较的次数极小化，故可以假定没有

做多余的比较；因此有一个扩展的二叉树结构，其中每个外部节点都对应一个排列。输入键码的所有排列都是可能的，而且每个排列都定义从根到外部节点的惟一路径；由此推知，在一株对 n 个元素进行比较而没有多余比较的比较树中，恰有 $n!$ 个外部节点。

最好的最坏情况 自然出现的第一个问题，是找出使所做的最大比较次数为极小的比较树(后面将考虑平均的比较次数)。

令 $S(n)$为足以将 n 个元素排序的极小比较次数。如果一个比较树的所有内部节点都在$<k$ 的层次内，则显然在这树中至多可以有 2^k 个外部节点。因此，令 $k = S(n)$，就有

$$n! \leqslant 2^{S(n)}$$

由于 $S(n)$是一个整数，我们可以重写这个公式以得到下界

$$S(n) \geqslant \lceil \lg n! \rceil \tag{1}$$

注意，按斯特林近似公式

$$\lceil \lg n! \rceil = n\lg n - n/\ln 2 + \frac{1}{2}\lg n + O(1) \tag{2}$$

所以大约需要 $n\lg n$ 次比较。

关系(1)通常称为信息论下限，因为信息论专家将说在排序过程中获得了 $\lg n!$ 个“比特的信息”；每次比较至多产生一个“比特的信息”。像图 34 那样的树也称做“征询表”，它们的数学性质在 Claude Picard 的书*Théorie des Questionnaires* (Paris: Gauthier-Villars, 1965)中已进行过探讨。

在我们所见过的所有排序方法中，需要最少比较的 3 个方法是：二叉插入(参见 5.2.1小节)，树选择(参见 5.2.3 小节)，以及直接两路合并(参见算法 5.2.4L)。由习题1.2.4-42容易看出，二叉插入的极大比较次数是

$$B(n) = \sum_{k=1}^{n} \lceil \lg k \rceil = n\lceil \lg n \rceil - 2^{\lceil \lg n \rceil} + 1 \tag{3}$$

而在习题 5.2.4-14 中给出了两路合并中的极大比较次数。在 5.3.3 小节中，我们将看到，取决于树是怎样建立的，树选择或者有和二叉插入相同的比较的上限，或者有和两路合并相同的比较的上限。在所有 3 种情况下，都达到了 $n\lg n$ 的渐近值；把 $S(n)$的下限和上限结合起来，就证明了

$$\lim_{n\to\infty} \frac{S(n)}{n \lg n} = 1 \tag{4}$$

于是我们有了一个 $S(n)$的近似公式，但是还希望得到更精确的信息。下表给出了对于小的 n，上述量的准确值。

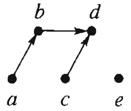
$n=$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$\lceil \ln n! \rceil=$	0	1	3	5	7	10	13	16	19	22	26	29	33	37	41	45	49
$B(n)=$	0	1	3	5	8	11	14	17	21	25	29	33	37	41	45	49	54

$L(n) = 0\ \ 1\ \ 3\ \ 5\ \ 9\ \ 11\ \ 14\ \ 17\ \ 25\ \ 27\ \ 30\ \ 33\ \ 38\ \ 41\ \ 45\ \ 49\ \ 65$

这里 $B(n)$ 和 $L(n)$ 分别对应于二叉插入和二路表合并。对所有的 n，可以证明 $B(n) \leqslant L(n)$（见习题 2）。

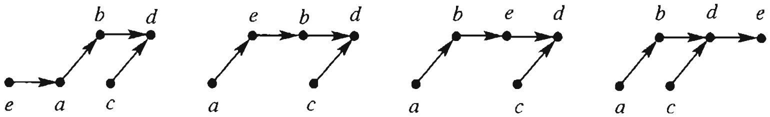
从上表可以看出 $S(4)=5$，但 $S(5)$ 可以是 7 或 8。这把我们带回到 5.2 节开始处所叙述的一个问题：什么是对 5 个元素进行排序的最好方法？能否仅仅使用 7 次比较，对 5 个元素进行排序？

回答是肯定的，但找出这 7 步过程并不特别容易。开始时，就像用合并对 4 个元素排序一样，首先比较 $K_1 : K_2$，接着 $K_3 : K_4$，然后把每对的较大者拿来比较，这就产生了一个可以被表示为



(5)

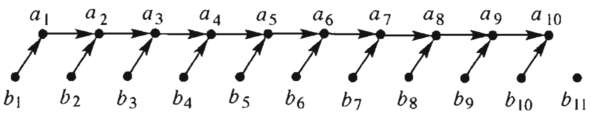
的图式，以示 $a<b<d$ 和 $c<d$（用这种有向图来表示元素之间的已知次序关系，是方便的。当且仅当在该有向图中有一条从 x 到 y 的路径时 x 小于 y）。这时，我们把第 5 个元素 $K_5 = e$ 插入到 $\{a, b, d\}$ 当中的适当位置，只需要比较两次，因为可以首先同 b 进行比较，而后同 a 或 d 进行比较，这有下列 4 种可能：



(6)

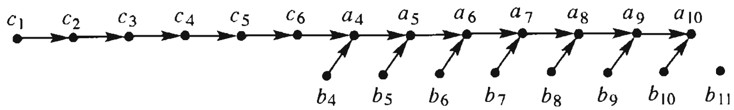
而且在每种情况下，都可以以另两次比较，把 c 插入小于 d 的剩下的元素当中。H. B. Demuth 首先发现了对 5 个元素排序的这两种方法[Ph. D. thesis, Stanford University(1956), 41～43]。

合并插入　Lester Ford, Jr 和 Selmer Johnson 发现了上述方法的一个令人满意的推广。由于它包含了合并及插入的各一部分，所以我们称它为合并插入。例如，考虑对 21 个元素的排序问题。开始先比较 10 对 $K_1 : K_2, K_3 : K_4, \cdots, K_{19} : K_{20}$，然后对这些对中 10 个较大元素进行排序，并利用合并插入。结果，得到类似于(5)的图形

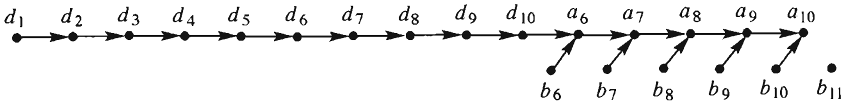


(7)

下一步是在 $\{b_1, a_1, a_2\}$ 中插入 b_3，然后把 b_2 插入到小于 a_2 的其它元素当中；得到图形，上边一行的元素称为主链。利用 3 次比较（首先 b_5 同 c_4 进行比较，然后与 c_2 或 c_6 比，等等）可以把 b_5 插入到主链中的适当位置。然后 b_4 可以用另外 3 个步骤移入主链中，结果得到(9)。



(8)



(9)

下一步是关键的；知道该做什么吗？仅仅利用 4 次比较，我们就把 b_{11}（而不是 b_7）插入到主链。然后，同样把 b_{10},b_9,b_8,b_7,b_6（以这个次序）插入到主链中的适当位置，每个至多使用 4 次比较。

这里涉及的对比较次数的仔细计算，说明 21 个元素至多用 $10+S(10)+2+2+3+3+4+4+4+4+4+4=66$ 步即可排序。由于

$$2^{65} < 21! < 2^{66}$$

我们也知道，在任何情况下，少于 66 次将是不可能的；因此

$$S(21) = 66 \tag{10}$$

（二叉插入要求 74 次比较）。

一般地说，n 个元素的合并插入按如下方式进行：

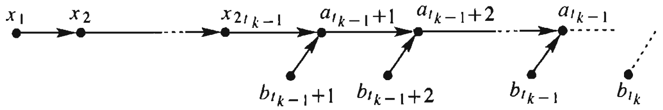
i）对于 $\lfloor n/2 \rfloor$ 个不相交的元素对，进行逐对比较（如果 n 为奇数，则剩下一个）。

ii）通过合并插入，将步骤 i）中找到的 $\lfloor n/2 \rfloor$ 个较大的数进行排序。

iii）像在（7）中那样命名元素 $a_1,a_2,\cdots a_{\lfloor n/2 \rfloor},b_1,b_2,\cdots b_{\lceil n/2 \rceil}$，其中 $a_1 \leqslant a_2 \leqslant \cdots \leqslant a_{\lfloor n/2 \rfloor}$ 且对于 $1 \leqslant i \leqslant \lfloor n/2 \rfloor$，$b_i \leqslant a_i$；称 b_1 和诸 a 为“主键码”。以下列次序用二叉插入把诸 b_j 插入到主链中，其中 $j > \lceil n/2 \rceil$：

$$b_3,b_2;\ b_5,b_4;\ b_{11},b_{10},\cdots,b_6;\ \cdots;\ b_{t_k},b_{t_k-1},\cdots,b_{t_{k-1}+1};\ \cdots \tag{11}$$

我们希望定义序列 $(t_1,t_2,t_3,t_4,\cdots)=(1,3,5,11,\cdots)$，它以这样一种方式出现于（11）中，即 $b_{t_k},b_{t_k-1},\cdots,b_{t_{k-1}+1}$ 中的每一个都可以用至多 k 次比较插入到主链中。推广（7），（8）和（9），我们得到图形



其中达到并包括 a_{t_k-1} 的主链包含 $2t_{k-1}+(t_k-t_{k-1}-1)$ 个元素。这个数必然小于 2^k；我们最好的办法是把它置成等于 2^k-1，使得

$$t_{k-1} + t_k = 2^k \tag{12}$$

由于 $t_1=1$，为了方便起见我们可以置 $t_0=1$，通过对一个几何级数求和得到

$$t_k = 2^k - t_{k-1} = 2^k - 2^{k-1} + t_{k-2} = \cdots =$$

$$2^k - 2^{k-1} + \cdots + (-1)^k 2^0 = (2^{k+1} + (-1)^k)/3 \tag{13}$$

(奇怪的是,这同一序列出现在我们为计算两个整数的最大公因子的一个算法分析中,参见习题 4.5.2-36。)

令 $F(n)$是通过合并插入将 n 个元素排序所需要的比较次数,显然,

$$F(n) = \lfloor n/2 \rfloor + F(\lfloor n/2 \rfloor) + G(\lceil n/2 \rceil) \tag{14}$$

其中 G 表示包含在步骤 iii)中的工作量。如果 $t_{k-1} \leqslant m \leqslant t_k$,则通过部分求和我们有

$$G(m) = \sum_{j=1}^{k-1} j(t_j - t_{j-1}) + k(m - t_{k-1}) = km - (t_0 + t_1 + \cdots + t_{k-1}) \tag{15}$$

置

$$w_k = t_0 + t_1 + \cdots + t_{k-1} = \lfloor 2^{k+1}/3 \rfloor \tag{16}$$

使得$(w_0, w_1, w_2, w_3, w_4, \cdots) = (0,1,2,5,10,21,\cdots)$。习题 13 证明

$$F(n) - F(n-1) = k \quad \text{当且仅当} \quad w_k < n \leqslant w_{k+1} \tag{17}$$

并且后一条件等价于

$$\frac{2^{k+1}}{3} < n \leqslant \frac{2^{k+2}}{3}$$

或者

$$k + 1 < \lg 3n \leqslant k + 2$$

因此

$$F(n) - F(n-1) = \left\lceil \lg \frac{3}{4} n \right\rceil \tag{18}$$

(这个公式来自 A. Hadian)[Ph. D. thesis, Univ. of Minnesota(1969), 38~42])。由此得出,$F(n)$有相当简单的表达式

$$F(n) = \sum_{k=1}^{n} \left\lceil \lg \frac{3}{4} k \right\rceil \tag{19}$$

它十分类似于二叉插入的对应公式(3)。这个和的"封闭形式"出现于习题 14 中。

利用等式(19)不难构造 $F(n)$的一个表;我们有

$n =$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$\lceil \lg n! \rceil =$	0	1	3	5	7	10	13	16	19	22	26	29	33	37	41	45	49
$F(n) =$	0	1	3	5	7	10	13	16	19	22	26	30	34	38	42	46	50

$n =$	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
$\lceil \lg n! \rceil =$	53	57	62	66	70	75	80	84	89	94	98	103	108	113	118	123
$F(n) =$	54	58	62	66	71	76	81	86	91	96	101	106	111	116	121	126

注意对于 $1 \leqslant n \leqslant 11$ 和 $20 \leqslant n \leqslant 21$,$F(n) = \lceil \lg n! \rceil$,所以我们知道对于这些 n,合并插入是最优的:

$$S(n) = \lceil \lg n! \rceil = F(n) \quad \text{对于 } n = 1, \cdots, 11, 20 \text{ 和 } 21 \tag{20}$$

Hugo Steinhaus 在他的精典著作(*Mathematical Snapshots* (Oxford University

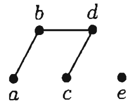
Press, 1950)第 2 版的 38～39 页中提出了求 $S(n)$的问题。他描述了二叉插入的方法——对 n 个对象排序的最好方法是:先将前 $n-1$ 个排好序,再考虑第 n 个;并且,他还推测,一般情况下,二叉插入是最优的。几年后[*Calcutta Math. Soc. Golden Jubilee Commemoration* **2**(1959),323～327]他做报告时说,他的两个同事 S. Trybula)和 P. Czen,"最近"已否定了他的推测,并且已经确定了 $n \leqslant 11$ 时的 $S(n)$。Trybula 和 P. Czen 可能已经独立地发现了合并插入的方法,但发表它的却是不久后 Ford 和 Johnson[*AMM* **66**(1959),387～389]。

在发现了合并插入之后,第一个未知的 $S(n)$值是 $S(12)$。表 1 显示 12! 十分接近于 2^{29},所以不大可能只用 29 步对 12 个元素进行排序。因此 Mark Wells 进行了一次穷举查找(在一台 Maniac Ⅱ 计算机上大约花了 60h),他发现 $S(12)=30$ [*Proc. IFIP Congress 65* **2**(1965),497～498;*Elements of Combinatorial* (Pergamon, 1971),213～215]。于是证明合并插入过程对于 $n=12$ 也是最优的。

表 1 阶乘值的二进制表示

$(1)_2 = 1!$
$(10)_2 = 2!$
$(110)_2 = 3!$
$(11000)_2 = 4!$
$(1111000)_2 = 5!$
$(1011010000)_2 = 6!$
$(1001110110000)_2 = 7!$
$(1001110110000000)_2 = 8!$
$(1011000100110000000)_2 = 9!$
$(1101110101111100000000)_2 = 10!$
$(10011000010001010100000000)_2 = 11!$
$(11100100011001111110000000000)_2 = 12!$
$(1011100110010100011001100000000000)_2 = 13!$
$(1010001001100001110110010100000000000)_2 = 14!$
$(10011000001110111011101110101100000000000)_2 = 15!$
$(100110000011101110111011101011000000000000000)_2 = 16!$
$(10100001101111111011101110110011011000000000000000)_2 = 17!$
$(101101011111011101100110010100111001100000000000000000)_2 = 18!$
$(1101100000001010111001001100000011010001001000000000000000000)_2 = 19!$
$(10000111000011011001110111111001000000101011010000000000000000000000)_2 = 20!$

***一个较深入的分析** 为了更仔细地研究 $S(n)$,我们更严密地考察例如(5)那样的偏序图式。在进行了若干次比较之后,可以借助于一个有向图来表示已经获得的知识。鉴于<关系的传递性,这个有向图不包含回路,所以可以把所有的有向边都画成从左到右;因此不妨去掉图中的箭头。于是(5)变为



(21)

如果 G 是这样一个有向图,令 $T(G)$是同 G 一致的排列的数目,即,把整数{1,2,

$\cdots,n\}$ 赋予 G 各顶点的方式种数，使得在 G 中每当有 $x\to y$ 时，顶点 x 上的数均小于顶点 y 上的数。例如，同(21)一致的一种排列是 $a=1,b=4,c=2,d=5,e=3$。在 5.1.4 小节中已经对各种 G 研究了 $T(G)$，其中我们发现，$T(G)$ 表示 G 可以被拓扑地排序的方式种数。

如果 G 是在进行了 k 次比较之后可以得到的 n 个元素的一个图，我们定义 G 的效率为

$$E(G)=\frac{n!}{2^kT(G)} \tag{22}$$

(这个思想来自于 Frank Hwang 和 Shen Lin)。严格地说，效率并不单独是图 G 的一个函数，它依赖于一个排序过程中得到 G 的方式，但是在叙述时不妨略去这一点。在元素 i 和 j 之间进行了另一次比较之后，得到两个图 G_1 和 G_2，一个是对于情况 $K_i<K_j$ 的，一个是对于情况 $K_i>K_j$ 的。显然

$$T(G)=T(G_1)+T(G_2)$$

如果 $T(G_1)\geqslant T(G_2)$，则有

$$T(G)\leqslant 2T(G_1)$$

$$E(G_1)=\frac{n!}{2^{k+1}T(G_1)}=\frac{E(G)T(G)}{2T(G_1)}\leqslant E(G) \tag{23}$$

因此，每次比较导致了一个具有更小或相等效率的图；不可能通过做进一步的比较来改进效率。

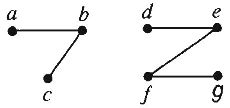
当 G 全然没有有向边时，有 $k=0$ 及 $T(G)=n!$，所以初始的效率是 1。另一个极端，当 G 是表示排序最后结果的一个图时，G 看起来像一条直线，且 $T(G)=1$。于是，例如，如果要寻找一个排序过程，在 7 步或更少的步骤内对 5 个元素排序，则必须得到其效率为 $5!/(2^7\times 1)=120/128=15/16$ 的线性图 •—•—•—•—•。由此得出在排序过程中出现的所有图必然有 $\geqslant\frac{15}{16}$ 的效率；如果出现任何低效率的图，则至少它的后裔之一也将是低效率的，而且最终将达到其效率 $<\frac{15}{16}$ 的一个线性图。一般地说，这个论证证明，所有对应于 n 个元素排序过程的树节点的图，都必然有 $\geqslant n!/2^l$ 的效率，其中 l 是树的层数。这是证明 $S(n)\geqslant\lceil\lg n!\rceil$ 的另一个方法，尽管这个论证同我们前面所说的实际上并没有多少不同。

图(21)的效率为 1，因为 $T(G)=15$，并且 G 已经在 3 次比较中得到。为了看看下次应当比较什么顶点，可以形成比较矩阵

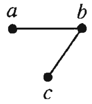
$$C(G)=\begin{matrix} & a & b & c & d & e \\ a & 0 & 15 & 10 & 15 & 11 \\ b & 0 & 0 & 5 & 15 & 9 \\ c & 5 & 10 & 0 & 15 & 9 \\ d & 0 & 0 & 0 & 0 & 3 \\ e & 4 & 8 & 6 & 12 & 0 \end{matrix} \tag{24}$$

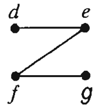
其中 C_{ij} 是通过把弧 $i \to j$ 加到 G 得到的图 G_1 的 $T(G_1)$。例如，如果比较 K_c 和 K_e，则同 G 相一致的 15 个排列分裂成 $K_e < K_c$ 的 $C_{ec} = 6$ 个和 $K_c < K_e$ 的 $C_{ce} = 9$ 个。后一个图的效率是 $15/(2 \times 9) = \frac{5}{6} < \frac{15}{16}$，所以它不可能导致一个 7 步排序过程。下一个比较必然是 $K_b : K_e$，以便保持 $\geqslant \frac{15}{16}$ 的效率。

当考虑图的连通分量时，效率的概念是特别有用的。例如考虑图

$G =$ 

它有两个分量

$G' =$ 

和 $G'' =$ 

由于没有连接 G' 和 G'' 的有向边，所以它由完全在 G' 之内进行的某些比较和完全在 G'' 之内进行的其它比较形成。一般地说，假定 $G = G' \oplus G''$ 没有 G' 和 G'' 之间的有向边，其中 G' 和 G'' 分别有 n' 和 n'' 个顶点，容易得出

$$T(G) = \binom{n' + n''}{n'} T(G') T(G'') \tag{25}$$

因为 G 的每个一致的排列都通过选择 n' 个元素赋给 G'，然后独立地在 G' 和 G'' 之内构造一致的排列而得到。如果在 G' 内已经进行了 k' 次比较，在 G'' 内进行了 k'' 次比较，则有基本的结果

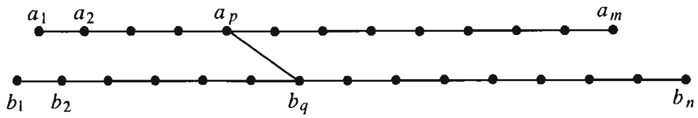
$$E(G) = \frac{(n' + n'')!}{2^{k'+k''} T(G)} = \frac{n'!}{2^{k'} T(G')} \cdot \frac{n''!}{2^{k''} T(G'')} = E(G')E(G'') \tag{26}$$

说明一个图的效率以一种简单的方式同其分量的效率有关。因此，我们可以限于考虑仅有一个分量的图。

现在假定 G' 和 G'' 是一个分量的图，并且假设要通过比较 G' 的一个顶点 x 和 G'' 的一个顶点 y 而把它们钩到一起。要知道这个效率是什么，为此，需要一个可以通过

$$\begin{pmatrix} p & & q \\ & < & \\ m & & m \end{pmatrix} \tag{27}$$

表示的函数，定义为同图



(28)

一致的排列的数目。于是$\left({p \atop m} < {q \atop n}\right)$等于$\binom{m+n}{m}$乘以一个概率数，该概率数是一个 m 个数集合的第 p 个最小者，小于独立地选择的 n 个数集合的第 q 个最小者的概率。习题 17 证明，可以借助于二项式系数以两种方式表达$\left({p \atop m} < {q \atop n}\right)$

$$\left({p \atop m} < {q \atop n}\right) = \sum_{0 \leqslant k < q} \binom{m-p+n-k}{m-p}\binom{p-1+k}{p-1} = \sum_{p \leqslant j \leqslant m} \binom{n-q+m-j}{n-q}\binom{q-1+j}{q-1} \tag{29}$$

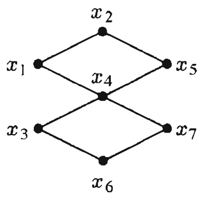
（顺便说一句，从代数上看来，如下一件事决不是明显的：这些二项式系数乘积的两个和居然是相等的）。我们也有公式

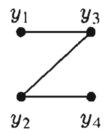
$$\left({p \atop m} < {q \atop n}\right) + \left({q \atop n} < {p \atop m}\right) = \binom{m+n}{m} \tag{30}$$

$$\left({q \atop n} < {p \atop m}\right) = \left({m+1-p \atop m} < {n+1-q \atop n}\right) \tag{31}$$

$$\left({p \atop m} < {q \atop n}\right) = \left({p \atop m-1} < {q \atop n}\right) + \left({p \atop m} < {q \atop n-1}\right) + [p \leqslant m][q = n]\binom{m+n-1}{m} \tag{32}$$

为了确定起见，现在考虑两个图

$G' =$ 

$G'' =$ 

(33)

通过直接枚举，不难证明 $T(G')=42$ 和 $T(G'')=5$；所以如果 G 是以 G'和 G''作为分量的 11 个顶点的图，则按等式(25)，我们有 $T(G)=\binom{11}{4}\cdot 42\cdot 5=69300$。如果要知道对于每个 i 和 j 有多少 $x_i<y_j$，这是待列出的惊人的排列数目。但是，这个计算可以在不到一个小时之内用手算出，如下：构造矩阵 $A(G')$和 $A(G'')$，其中 A_{ik}是使得 x_i（或 y_i）等于 k 的 G'（或 G''）的一致排列的个数。于是，其中 x_i 小于 y_i 的 G 的排列个数为 $A(G')$的(i,p)元素乘以$\left({p \atop 7} < {q \atop 4}\right)$再乘以 $A(G'')$的(j,q)元素，对 $1\leqslant p\leqslant 7$ 和 $1\leqslant q\leqslant 4$ 进行求和。换言之，我们要构造矩阵乘积 $A(G')\cdot L\cdot A(G'')^T$，其中 $L_{pq}=\left({p \atop 7} < {q \atop 4}\right)$。这得到

$$\begin{pmatrix}21&16&5&0&0&0&0\\0&5&10&12&10&5&0\\21&16&5&0&0&0&0\\0&0&12&18&12&0&0\\0&0&0&0&5&16&21\\0&5&10&12&10&5&0\\0&0&0&0&5&16&21\end{pmatrix}\begin{pmatrix}210&294&322&329\\126&238&301&325\\70&175&265&315\\35&115&215&295\\15&65&155&260\\5&29&92&204\\1&8&36&120\end{pmatrix}\begin{pmatrix}2&3&0&0\\2&2&0&1\\1&0&2&2\\0&0&3&2\end{pmatrix}=$$

$$\begin{pmatrix}48169&42042&66858&64031\\22825&16005&53295&46475\\48169&42042&66858&64031\\22110&14850&54450&47190\\5269&2442&27258&21131\\22825&16005&53295&46475\\5269&2442&27258&21131\end{pmatrix}$$

于是钩起 G' 和 G'' 的“最好”方法是把 x_1 同 y_2 加以比较；当 $x_1<y_2$ 时有 42042 种情况，当 $x_1>y_2$ 时有 $69300-42042=27258$ 种情况（由对称性，我们也可以比较 x_3 和 y_2，x_5 和 y_3，或 x_7 和 y_3，导出实际上相同的结果）。对于 $x_1<y_2$ 得到的图效率为

$$\frac{69300}{84084}E(G')E(G'')$$

它不是太好的；因此，在任何排序方法中把 G' 和 G'' 钩起来大概是一种坏的想法！这个例子的要点是，无须多余的计算就能够做出这样一个判断来。

这些思想可用来独立地验证 Mark Wells 关于 $S(12)=30$ 的证明。从包含一个顶点的图开始，可以重复地尝试用这样一种方式对图 G 之一或对 $G'\oplus G''$（一对图分量 G' 和 G''）增加一个比较，使得两个由此得到的图只有 12 个或更少的顶点，且效率 $\geqslant 12!/2^{39}\approx 0.89221$。只要这是可能的，就取所得到的最低效率的图，并且把它加到我们的集合中，除非两个图之一同构于集合中已有的一个图。如果得到的两个图有同样的效率，则就任意地选择它们之一。一个图可以由它的对偶（通过颠倒次序得到）来标识，只要我们考虑既把比较加到 $G'\oplus G''$ 上，也加到 $G'\oplus$ 对偶 (G'') 上。图 36 所示为以这一方式得到的一些最小的图及它们的效率。

在这个过程结束之前，由计算机恰好生成了 1649 个图。由于得不到图
，故可以得出结论，$S(12)>29$。看来可以通过类似实验，花费不太长的时间以推导出 $S(22)>70$，因为 $22!/2^{70}\approx 0.952$ 表明必须有极高的效率才能在 70 步之内进行排序（在用 12 个或更少的顶点建立的 1649 个图中，仅有 91 个有这样的高效率）。

Marcin Peczarski[*Lecture Notes in Comp. Sci.* **2461**(2002)，785～794]扩展了 Wells 的方法，并证明 $S(13)=34$；因此在这种情况下合并插入也是最优的。直觉地，似乎总有一天能够证明 $S(16)<F(16)$，因为比起通过 $S(10)$ 次比较对 10 个元素排序，然后用二叉插入逐个插入其它 6 个元素来，$F(16)$ 所需的比较次数决不会更少。必然存在一种办法来改进这样一种方法！但是现在，确定地知道 $F(n)$ 是非最优的最小情况是

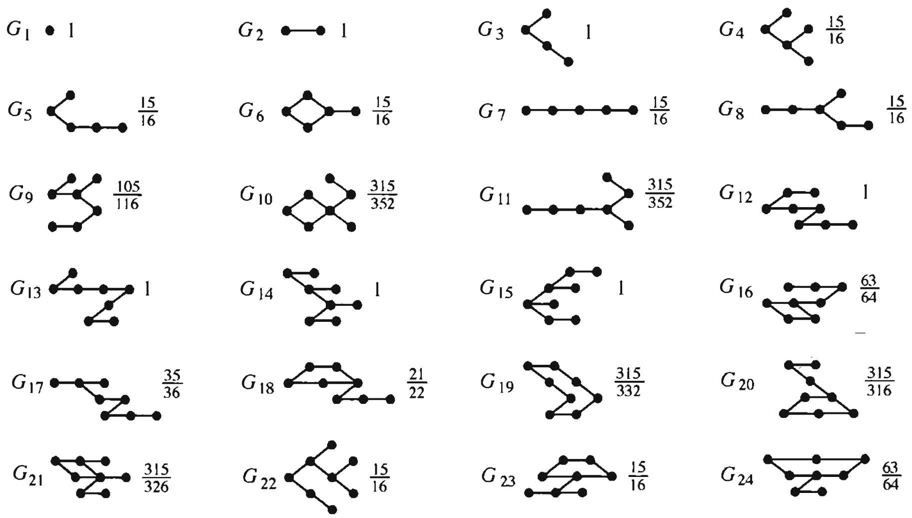


图 36　在 $S(12)>29$ 的一个长证明的最初几步中，得到的一些图和它们的效率

$n=47$：在通过 $F(5)+F(42)=178$ 次比较对 5 个和 42 个元素进行排序之后，使用由 J. Schulte Mönting[*Theoretical Comp. Sci.* **14**(1981)，19～37]给出的方法，我们可以以 22 个进一步的比较，来合并这些结果；这就击败了 $F(47)=201$(Glenn K. Manacher[*JACM* **26** (1979)，441～456]以前已经证明，由 $n=189$ 开始，有无穷多个 n 存在，使 $S(n)<F(n)$)。

比较的平均次数　至今已经考虑了一些过程，在它们的最坏情形尚可这个意义下它们是最好的；换句话说，已经寻道过“极小化极大”的过程，对于比较的极大次数而言它取极小。现在让我们寻找一个“极小化平均值”的过程，它对比较的平均次数取极小，假定输入是随机的，于是每种排列都是同等可能的。

如图 34 所示，再次考虑一个排序过程的树表示。对所有排列取平均，在该树中的平均比较次数是

$$\frac{2+3+3+3+3+2}{6}=2\frac{2}{3}$$

一般地说，在一个排序方法中，平均的比较次数是*树的外部路径长度除以 $n!$*（回想一下，外部路径长度是从根到每个外部节点的距离之和；见 2.3.4.5 小节）。从 2.3.4.5 小节的一些考虑容易看出，如果一个具有 N 个外部节点的二叉树在 $q-1$ 层有 2^q-N 个外部节点，而在 q 层有 $2N-2^q$ 个节点，其中 $q=\lceil \lg N\rceil$(根在 0 层)，则此树具有极小的外部路径长度。因此极小的外部路径长度是

$$(q-1)(2^q-N)+q(2N-2^q)=(q+1)N-2^q \tag{34}$$

极小路径长度也能由另一种有趣的方式加以表征：*当且仅当有一个数 l，使得一株扩展的二叉树的所有外部节点都出现在 l 层和 $l+1$ 层上时，此树有极小的外部路径长度*(见习题 20)。

如果置 $q=\lg N+\theta$,其中 $0\leqslant\theta<1$,则极小外部路径长度的公式变为:

$$N(\lg N+1+\theta-2^{\theta}) \tag{35}$$

函数 $1+\theta-2^{\theta}$ 如图 37 所示;对于 $0<\theta<1$,它虽是正的但非常小,决不超过

$$1-(1+\ln\ln 2)/\ln 2=0.08607\ 13320\ 55934+ \tag{36}$$

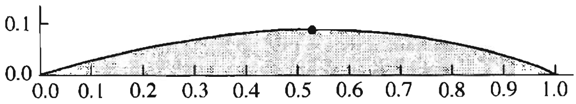


图 37　函数 $1+\theta-2^{\theta}$

于是,通过将(35)除以 N 得到的极小可能平均路径长度,决不小于 $\lg N$ 也决不大于 $\lg N+0.0861$(这个结果首先由 A. Gleason 在 1956 年得出)。

现在如果置 $N=n!$,则得到在任何排序方案中平均比较次数的下限。渐近地说,这个下限是

$$\lg n!+O(1)=n\lg n-n/\ln 2+O(\log n) \tag{37}$$

设 $\bar{F}(n)$是由合并插入算法执行的平均比较次数;我们有

$n=$	1	2	3	4	5	6	7	8
下限(34) =	0	2	16	112	832	6896	62368	619904
$n!\bar{F}(n)=$	0	2	16	112	832	6912	62784	623232

于是对于 $n\leqslant 5$ 合并插入在两种意义下都是最优的,但是对于 $n=6$,它平均有 6912/720 = 9.6 次比较,而我们的下限表明平均 6896/720 = 9.577777…次比较是可能的。片刻的思考就可明白为什么这是正确的:在仅仅做了 8 次比较之后采用合并插入法,即可把 6 个元素的某些“幸运的”排列排好序,所以比较树在三个层次上而不是在两个层次上出现外部节点。这使得整个路径长度变长了。习题 24 说明有可能构造一个 6 元素的排序过程。这个过程在每种情况下都要求 9 或 10 次比较;由此得出,平均来说,这个方法比合并插入优越。而且在其最坏情况下,也不比合并插入更坏。

当 $n=7$ 时,Y. Césari[Thesis (Univ. of Paris, 1968), Page 37]已经说明,没有排序方法能达到外部路径长度的下界 62368(利用习题 22 的结果,有可能不用计算机来证明这个事实)。另一方面,他已经构造了一些过程,当 $n=9$ 或 10 时,它们达到下限(34)。一般地说,使平均比较次数极小化的问题,实质上比确定 $S(n)$的问题更为困难。甚至可能对于某个 n,所有使平均比较次数极小化的方法在它们最坏的情况下要求进行 $S(n)$次以上的比较。

习　题

1. [*20*] 分别用以下两种方法画出对 4 个元素进行排序的比较树:(a)二叉插入法;(b)直接的两路合并法。这些树的外部路径长度是多少?

2.[*M24*] 证明 $B(n) \leqslant L(n)$，并求使等式成立的所有的 n。

3.[*M22*] 当允许存在相等的键码时，在对 3 个元素排序时有 13 种可能的结果：

$$\begin{array}{lll} K_1 = K_2 = K_3 & K_1 = K_2 < K_3 & K_1 = K_3 < K_2 \\ K_2 = K_3 < K_1 & K_1 < K_2 = K_3 & K_2 < K_1 = K_3 \\ K_3 < K_1 = K_2 & K_1 < K_2 < K_3 & K_1 < K_3 < K_2 \\ K_2 < K_1 < K_3 & K_2 < K_3 < K_1 & K_3 < K_1 < K_2 \\ K_3 < K_2 < K_1 & & \end{array}$$

令 P_n 表示在对 n 个元素排序，而且允许存在相等项时可能的结果数目，使得 $(P_0, P_1, P_2, P_3, P_4, P_5, \cdots) = (1, 1, 3, 13, 75, 541, \cdots)$。证明生成函数 $P(z) = \sum_{n \geqslant 0} P_n z^n / n!$ 等于 $1/(2 - e^z)$。提示：证明

$$P_n = \sum_{k > 0} \binom{n}{k} P_{n-k} \quad 当\ n > 0$$

4.[*HM27*](O.A.Gross) 确定当 $n \to \infty$ 时，习题 3 的数 P_n 的渐近值[可能的提示：考虑 $\cot z$ 的部分分式展开]。

5.[*16*] 当键码可以相等时，每个比较可以有 3 种(而不是 2 种)结果：$K_i < K_j, K_i = K_j, K_i > K_j$。对于这种一般情况下的排序算法，可用扩展的三叉树表示，其中每个内部节点 $i : j$ 有 3 株子树；左边、中间和右边子树分别对应于比较的 3 种可能的结果。

试画出一株扩展的三叉树，它定义了允许有相等键码的 $n = 3$ 时的排序算法。这株树应有 13 个外部节点，对应于习题 3 中列出的 13 种可能的结果。

▶6.[*M22*] 设每个比较如习题 5 中那样有 3 种结果，$S'(n)$ 是对 n 个元素排序并确定诸键码之间所有等式所需要的最小比较数。正文中的"信息论"论证方法可以很容易地被推广，以证明 $S'(n) \geqslant \lceil \log_3 P_n \rceil$，其中 P_n 是习题 3 和 4 中所分析的函数；但请证明，事实上 $S'(n) = S(n)$。

7.[*20*] 在习题 5 对 4 个元素排序的意义下，当已知所有键码都是 0 或 1 时(即，如果 $K_1 < K_2$ 和 $K_3 < K_4$，则有 $K_1 = K_3$ 和 $K_2 = K_4$)，画出扩展的三叉树。假定 2^4 种输入都是同等可能的，请使用极小的平均比较次数。注意找出所有存在的等式。例如，当你只知道 $K_1 \leqslant K_2 \leqslant K_3 \leqslant K_4$ 时切勿停止排序。

8.[*26*] 当已知所有的键码皆为 -1、0 或 $+1$ 时，像习题 7 中那样，对 4 个元素排序，画出一株扩展的三叉树。假定 3^4 种输入都是同等可能的，请使用极小的平均比较次数。

9.[*M20*] 当像在习题 7 中那样，对 n 个元素进行排序，且又知道所有键码皆为 0 或 1 时，试问在最坏情况下极小的比较次数是多少？

▶10.[*M25*] 当像在习题 7 中那样，对 n 个元素进行排序，且又知道所有键码皆为 0 或 1 时，试问作为 n 的函数，极小平均比较次数是多少？

11.[*HM27*] 当像在习题 5 中那样，对 n 个元素进行排序，且又知道所有键码都是集合 $\{1, 2, \cdots, m\}$ 上的数时，令 $S_m(n)$ 是在最坏的情况下需要的极小比较数[于是，由习题 6，$S_n(n) = S(n)$]，试证明对于固定的 m，当 $n \to \infty$ 时，$S_m(n)$ 渐近于 $n \lg m + O(1)$。

▶12.[*M25*] (W.G.Bouricius，约 1954)假定可以出现相等的键码，但是只需要对元素 $\{K_1, K_2, \cdots, K_n\}$ 排序，找出排列 $a_1 a_2 \cdots a_n$，使得 $K_{a_1} \leqslant K_{a_2} \leqslant \cdots K_{a_n}$；我们不需要知道 K_{a_i} 和 $K_{a_{i+1}}$ 是否相等。

如果一株比较树在上述意义下对一个键码序列排序，当 $K_i = K_j$ 时它在节点 $i : j$ 之下任取一分支(这株树是二叉的，而不是三叉的)，则这种排序称为强排序。

a)证明：没有多余比较的一个比较树对每个键码序列强排序，当且仅当它对每个无相同键码

的序列进行排序。

b)证明:当且仅当一株比较树对 0 和 1 的每个序列进行强排序时,它也对每个键码序列进行强排序。

13.[*M28*] 证明(17)。

14.[*M24*] 试求和数(19)的一个封闭形式。

15.[*M21*] 确定 $B(n)$和 $F(n)$的渐近特性,精确到 $O(\log n)$[提示:证明在这两种情况下,n 的系数都包含图 37 中所示的函数]。

16.[*HM26*] (F. Hwang 和 S. Lin) 证明当 $n \geqslant 22$ 时 $F(n) > \lceil \lg n! \rceil$。

17.[*M20*] 证明(29)。

18.[*20*] 如果过程的前些步如图 36 所示,并且它产生了具有效率 $12!/2^{29}$的线性图
,则这将证明 $S(12)=29$ 吗?

19.[*40*] 以下列带启发性的探索规则进行实验,来判定在设计一株比较树时下次应比较哪一对元素:在对$\{K_1,\cdots,K_n\}$进行排序的每一个阶段,对于 $1 \leqslant i \leqslant n$,令 u_i 是作为迄今所做比较的结果而得到的$\leqslant K_i$ 的键码数,v_i 是已知为$\geqslant K_i$ 的键码数。按递增的 u_i/v_i 对诸键码重新编号,使得 $u_1/v_1 \leqslant u_2/v_2 \cdots \leqslant u_n/v_n$。现在对于某个使$|u_i v_{i+1} - u_{i+1} v_i|$取极小的 i,比较 $K_i : K_{i+1}$(比起(24)中所用的完全比较矩阵,本方法基于少得多的信息,但它在许多情况下似乎给出最优的结果)。

▶20.[*M26*] 证明:当且仅当存在一个数 l,使得一株扩展的二叉树的所有外部节点都出现于第 l 层和第 $l+1$ 层上时(或许,所有的节点都在同一层上),该树有极小的外部路径长度。

21.[*M21*] 一株扩展的二叉树的高度是它的外部节点的极大层次数。如果 x 是一株扩展的二叉树的内部节点,令 $t(x)$是 x 以下的外部节点数,并令 $l(x)$表示 x 的左子树的根。如果 x 是外部节点,则令 $t(x)=1$。试证一株扩展二叉树在具有相同节点数的所有二叉数中具有极小高度,如果对于所有内部节点 x,

$$|t(x) - 2t(l(x))| \leqslant 2^{\lceil \lg t(x) \rceil} - t(x)$$

22.[*M24*] 继续习题 21,证明一株二叉树在具有同样节点数的所有二叉树当中,拥有极小外部路径长度的充分必要条件是对所有内部节点 x

$$|t(x) - 2t(l(x))| \leqslant 2^{\lceil \lg t(x) \rceil} - t(x) \text{ 和 } |t(x) - 2t(l(x))| \leqslant t(x) - 2^{\lfloor \lg t(x) \rfloor}$$

[例如,如果 $t(x)=67$,则我们必定有 $t(l(x))=32,33,34$ 或 35。如果只要求树的高度取极小,则由以前的习题,我们可以有 $3 \leqslant t(l(x)) \leqslant 64$]。

23.[*10*] 正文中证明,任何对 n 个元素进行排序的方法,其平均比较次数至少是$\lceil \lg n! \rceil \approx n\lg n$。但是多重表插入(程序 5.2.1M)平均仅仅花费 $O(n)$个时间单位。为什么能这样?

24.[*27*] (C. Picard) 试求一株使得所有外部节点都出现于 10 层和 11 层的对 6 个元素排序的树。

25.[*11*] 如果有一个对 7 个元素排序的过程,它达到等式(34)所预测的极小平均比较次数,则在 13 层上将有多少外部节点?

26.[*M42*] 试求对 7 个元素进行排序的一个过程,它的平均比较次数取极小。

▶27.[*20*] 假设已知配置 $K_1<K_2<K_3$,$K_1<K_3<K_2$,$<K_2<K_1<K_3$,$K_2<K_3<K_1$,$K_3<K_1<K_2$,$K_3<K_2<K_1$ 分别以概率 .01,.25,.01,.24,.25,.24 出现,试找出一株比较树,它以最小的平均比较次数对这 3 个元素排序。

28.[*40*] 写出一个 MIX 程序,它在最少时间内对 5 个单字长的键码进行排序,并停机(关于基本规则见 5.2 节开头)。

29.[*M25*] (S.M.Chase)　设 $a_1a_2\cdots a_n$ 是$\{1,2,\cdots,n\}$的一个排列。证明任何一个仅仅以诸 a 之间的比较为基础,来判断这个排列是偶还是奇(亦即它是否有偶的或奇的反序数)的算法,必须至少进行 $n\lg n$ 次比较,即使这个算法仅有两个可能的结果亦然。

30.[*M23*] (最优交换排序)　在 5.2.2 小节中定义的每一个交换排序算法都可表示为一株比较交换树;亦即一二叉树结构,其内部节点具有形式 $i:j, i<j$,它可解释为下列的操作:"如果 $K_i\leqslant K_j$,则通过取这株树的左分支继续进行;如果 $K_i>K_j$,则通过交换记录 i 和 j,而后取这株树的右分支继续进行"。当遇到一个外部节点时,$K_1\leqslant K_2\leqslant\cdots\leqslant K_n$ 必然为真。因此,一株比较交换树与一株比较树的区别在于:它既指明比较操作也指明数据移动。

令 $S_e(n)$表示在最坏的情况下,借助于一株比较交换树对 n 个元素排序所需要的极小比较交换数。证明 $S_e(n)\leqslant S(n)+n-1$。

31.[*M38*] 继续习题 30,证明 $S_e(5)=8$。

32.[*M42*] 继续习题 31,对于 $n>5$ 的小的值研究 $S_e(n)$。

33.[*M30*] (T.N.Hibbard)　阶为 x 和分辨度为 δ 的实值的查找树是一株扩展的二叉树,其中所有的节点都包含一个非负的实值,使得:(i)在每个外部节点中的值都$\leqslant\delta$;(ii)在每个内部节点的值至多为它的两个儿子值的和;(iii)根的值是 x。这样一株树的加权路径长度,定义为对于所有外部节点,节点的层次乘以它包含的值求和。

证明一株阶为 x 和分辨度为 1 的实值查找树,在有同样的阶和分辨度的所有这样的树中,具有极小的加权路径长度的充分必要条件是:在(ii)中的等式成立,且对于包含于兄弟节点中的所有值偶 x_0 和 x_1,下列进一步的条件成立:(iv)不存在整数 $k\geqslant 0$,使得 $x_0<2^k<x_1$ 或 $x_1<2^k<x_0$;(v)$\lceil x_0\rceil-x_0+\lceil x_1\rceil-x_1<1$(特别是,如果 x 是整数,则条件(v)意味着在这株树中的所有值都是整数,且条件(iv)等价于习题 22 的结果)。

另外证明对应的极小加权路径长度是 $x\lceil\lg x\rceil+\lceil x\rceil-2^{\lceil\lg x\rceil}$。

34.[*M50*] 对于无穷多个 n 确定 $S(n)$的精确值。

35.[*49*] 确定 $S(14)$的精确值。

36.[*M50*] (S.S.Kislitsyn,1968)证明或反驳:任何满足 $T(G)>1$ 的有向无回路图 G,有两个顶点 u 和 v,使得通过加上有向边 $u\leftarrow v$ 和 $u\rightarrow v$,从 G 得到的有向图 G_1 和 G_2 是无回路的,并满足 $1\leqslant T(G_1)/T(G_2)\leqslant 2$(因此,对于某个 u 和 v,$T(G_1)/T(G)$总介于$\frac{1}{3}$和$\frac{2}{3}$之间)。

*5.3.2　极少比较合并

现在考虑一个相关的问题:什么是把 m 个元素的一个有序集合和 n 个元素的一个有序集合合并的最好方法?以

$$A_1<A_2<\cdots<A_m \quad 和 \quad B_1<B_2<\cdots<B_n \tag{1}$$

表示有待合并的元素,如同在 5.3.1 小节中那样,假定这 $m+n$ 个元素是不同的。诸 A 可以$\binom{m+n}{m}$种方式出现于诸 B 当中,所以,已用于排序问题的论证立即告诉我们,至少需要

$$\left\lceil\lg\binom{m+n}{m}\right\rceil \tag{2}$$

次比较。如果置 $m=\alpha n$ 并设 $n\rightarrow\infty$,而 α 是固定的,则斯特林近似公式告诉我们

$$\lg\binom{\alpha n+n}{\alpha n}=n((1+\alpha)\lg(1+\alpha)-\alpha\lg\alpha)-\frac{1}{2}\lg n+O(1)\qquad(3)$$

正常的合并过程,即算法 5.2.4M,在它最坏的情况下,花费 $m+n-1$ 次比较。

令 $M(m,n)$ 表示类似于 $S(n)$ 的函数,亦即总是足以把 m 件事物同 n 个事物合并的极小比较次数。按我们刚才所做的观察

$$\left\lceil\lg\binom{m+n}{m}\right\rceil\leqslant M(m,n)\leqslant m+n-1\quad 对于所有\ m,n\geqslant 1\qquad(4)$$

公式(3)说明这个下限可以如何远离上限。当 $\alpha=1$(即 $m=n$)时,下限是 $2n-\frac{1}{2}\lg n+O(1)$,所以上下限两者都有正确数量级,但是它们之间的差可以任意大。当 $\alpha=0.5\left(即\ m=\frac{1}{2}n\right)$时,下限为

$$\frac{3}{2}n\left(\lg 3-\frac{2}{3}\right)+O(\log n)$$

它大约是 $\lg 3-\frac{2}{3}\approx 0.918$ 乘上限。而且,随着 α 减小,这些界限就越离越远,因为标准的合并算法主要是对 $m\approx n$ 的文件来设计的。

当 $m=n$ 时,合并的问题有相当简单的解;由此可知,是(4)的下界有问题,而不是上界。下列定理是由 R.L.Graham 和 R.M.Karp 大约于 1968 年独立地发现的。

定理 M *当 $m\geqslant 1$ 时,$M(m,m)=2m-1$。*

证明 考虑把 $A_1<\cdots<A_m$ 同 $B_1<\cdots<B_m$ 合并的任何算法。当它比较 $A_i:B_j$ 时,如果 $i<j$ 取分支 $A_i<B_j$;如果 $i\geqslant j$ 取分支 $A_i>B_j$。合并最终必须以配置

$$B_1<A_1<B_2<A_2<\cdots B_m<A_m\qquad(5)$$

结束,因为这同所采取的所有分支一致,而且 $2m-1$ 个比较 $B_1:A_1,A_1:B_2,B_2:A_2,\cdots,B_m:A_m$ 中的每一个都必须已经明显地做出,否则至少会有两个配置同已知的事实相一致。例如,如果 A_1 未与 B_2 进行比较,则配置

$$B_1<B_2<A_1<A_2\cdots<B_m<A_m$$

无法与(5)相区别。▌

简单修改这个证明,就得出伴随的公式

$$M(m,m+1)=2m\qquad 对于\ m\geqslant 0\qquad(6)$$

构造下限 定理 M 表明,“信息论”的下限(2)可以同真正的下限相距任意远;因此,用于证明定理 M 的技术给了我们发现下限的另一个办法,这样一个证明技术通常被看做是制造一个*敌手*,一个试图使这些算法缓慢地进行的有害的人。当用于合并的一个算法比较 $A_i:B_j$ 时,这个敌手是这样确定比较的结局的:就好像他要使这个算法沿着较为困难的道路走下去。如同在定理 M 的证明中那样,如果能够发明一个适当的敌手,就能够确保,每一个有效的合并算法必须做更大量的比较。

我们将利用受限的敌手,它在确定某些比较的结果时,其权力是受到限制的。在一个受限的敌手影响下的一个合并方法不知道这些限制,所以它仍然进行必要的比较,尽管比较的结果已被预先确定。例如,在关于定理 M 的证明中,通过条件(5)限制了所有的结果,然而合并算法不能利用这一事实来免去任何一次比较。

在以下的讨论中,我们将使用的限制可应用于文件的左端和右端。左边的限制以下列符号表征:

·意为没有左边的限制。

\ 意为所有的结论都必须同 $A_1 < B_1$ 一致。

/意为所有的结论都必须同 $A_1 > B_1$ 一致。

右边的限制以下列符号表征:

·意为没有右边的限制。

\ 意为所有的结论都必须同 $A_m < B_n$ 一致。

/意为所有的结论都必须同 $A_m > B_n$ 一致。

敌手有 9 种类型,以 $\lambda M\rho$ 来表示,其中 λ 是一个左限制而 ρ 是一个右限制。例如,一个"\ M \"敌手必须指出 $A_1 < B_j$ 和 $A_i < B_n$;一个". M ."敌手是不受限制的。对于某个小的 m 和 n,某些类型的受限制的敌手是不可能有的;当 $m=1$ 时,我们显然不能有一个"\ M/"的敌手。

现在让我们来构造用于合并的一个稍微复杂但非常难对付的敌手,它不总产生最优的结果,但它给出了包含大量有趣情况的下限。给定 m、n,以及左边和右边的限制 λ 和 ρ,假设要求敌手指出 A_i 或 B_j 哪一个大。有 6 个策略可用来把此问题归结为较小的 $m+n$ 的情况。

策略 A(k,l)　适用于 $i \leqslant k \leqslant m$ 以及 $1 \leqslant l \leqslant j$。比如说 $A_i < B_j$,而且要求随后的操作把 $\{A_1,\cdots,A_k\}$ 同 $\{B_1,\cdots,B_{l-1}\}$ 以及把 $\{A_{k+1},\cdots,A_m\}$ 同 $\{B_l,\cdots,B_n\}$ 合并。如果 $p \leqslant k$ 和 $q \geqslant l$,则比较 $A_p : B_q$ 所得的回答将为 $A_p < B_q$;如果 $p > k$ 和 $q < l$,则回答将是 $A_p > B_q$;如果 $p \leqslant k$ 和 $q < l$,则它们将通过$(k,l-1,\lambda,.)$敌手加以处理;如果 $p > k$ 和 $q \geqslant l$,则通过$(m-k,n+1-l,.,\rho)$敌手加以处理。

策略 B(k,l)　适用于 $i \leqslant k \leqslant m$ 和 $1 \leqslant l < j$。比如说 $A_i < B_j$,并且要求随后的操作把 $\{A_1,\cdots,A_k\}$ 同 $\{B_1,\cdots,B_l\}$,以及把 $\{A_{k+1},\cdots,A_m\}$ 同 $\{B_i,\cdots,B_n\}$ 加以合并,约定 $A_k < B_l < A_{k+1}$(注意,B_l 出现于有待合并的两个表中。条件 $A_k < B_l < A_{k+1}$ 确保合并一个组不会给出有助于合并另一个组的信息)。因此,将来的比较 $A_p : B_q$,如果 $p \leqslant k$ 和 $q \geqslant l$,将得到结果 $A_p < B_q$;如果 $p > k$ 和 $q \leqslant l$,结果将为 $A_p > B_q$;如果 $p \leqslant k$ 和 $q \leqslant l$,它们将通过一个$(k,l,\lambda,\backslash)$敌手来处理;如果 $p > k$ 和 $q \geqslant l$,则通过一个$(m-k,n+1-l,/,\rho)$ 敌手来处理。

策略 C(k,l)　适用于 $i < k \leqslant m$ 和 $1 \leqslant l \leqslant j$。比如说 $A_i < B_j$,并且要求随后的操作把 $\{A_1,\cdots,A_k\}$ 同 $\{B_1,\cdots,B_{l-1}\}$ 和 $\{A_k,\cdots,A_m\}$ 同 $\{B_l,\cdots,B_n\}$ 合并,约定 $B_{l-1} < A_k < B_l$(类似于策略 B,交换 A 和 B 的作用)。

策略 A′(k,l) 适用于 $1\leqslant k\leqslant i$ 和 $j\leqslant l\leqslant n$。比如说 $A_i>B_j$，并且要求把 $\{A_1,\cdots,A_{k-1}\}$ 同 $\{B_1,\cdots,B_l\}$，和 $\{A_k,\cdots,A_m\}$ 同 $\{B_{l+1},\cdots,B_n\}$ 合并(类似于策略 A)。

策略 B′(k,l) 适用于 $1\leqslant k\leqslant i$ 和 $j<l\leqslant n$。比如说 $A_i>B_j$，并且要求把 $\{A_i,\cdots,A_{k-1}\}$ 同 $\{B_1,\cdots,B_l\}$ 和 $\{A_k,\cdots,A_m\}$ 同 $\{B_l,\cdots,B_n\}$ 合并，且有 $A_{k-1}<B_l<A_k$(类似于策略 B)。

策略 C′(k,l) 适用于 $1\leqslant k<i$ 和 $j\leqslant l\leqslant n$。比如说 $A_i>B_j$，并且要求把 $\{A_1,\cdots,A_k\}$ 同 $\{B_1,\cdots,B_l\}$ 以及 $\{A_k,\cdots,A_m\}$ 同 $\{B_{l+1},\cdots,B_n\}$ 合并，且有 $B_l<A_k<B_{l+1}$(类似于策略 C)。

由于这些限制，上述策略不可能用于下列情况：

策略			当出现下列情况时必须被省略
$A(k,1)$,	$B(k,1)$,	$C(k,1)$	$\lambda=/$
$A'(1,l)$,	$B'(1,l)$,	$C'(1,l)$	$\lambda=\backslash$
$A(m,l)$,	$B(m,l)$,	$C(m,l)$	$\rho=/$
$A'(k,n)$,	$B'(k,n)$,	$C'(k,n)$	$\rho=\backslash$

令 $\lambda M\rho(m,n)$ 表示合并的极大下限，它是通过上述的某个敌手得到的。当第一个比较数是 $A_i:B_j$ 时，每个策略在其适用的场合给了我们关于这 9 个函数的一个不等式，如下：

$A(k,l)$: $\lambda M\rho(m,n)\geqslant 1+\lambda M.(k,l-1)+.M\rho(m-k,n+1-l)$;

$B(k,l)$: $\lambda M\rho(m,n)\geqslant 1+\lambda M\backslash(k,l)+/M\rho(m-k,n+1-l)$;

$C(k,l)$: $\lambda M\rho(m,n)\geqslant 1+\lambda M/(k,l-1)+\backslash M\rho(m+1-k,n+1-l)$;

$A'(k,l)$: $\lambda M\rho(m,n)\geqslant 1+\lambda M.(k-1,l)+.M\rho(m+1-k,n-l)$;

$B'(k,l)$: $\lambda M\rho(m,n)\geqslant 1+\lambda M\backslash(k-1,l)+/M\rho(m+1-k,n+1-l)$;

$C'(k,l)$: $\lambda M\rho(m,n)\geqslant 1+\lambda M/(k,l)+\backslash M\rho(m+1-k,n-l)$。

对于固定的 i 和 j，这个敌手将采取一个策略，该策略使得当 k 和 l 位于由 i 和 j 所允许的范围时，由所有可能的右端给出的下限取极大值；于是，定义 $\lambda M\rho(m,n)$ 为对于 $1\leqslant i\leqslant m$ 和 $1\leqslant j\leqslant n$ 所取的这些下限的极小值。当 m 或 n 为 0 时，$\lambda M\rho(m,n)$ 为 0。

例如，考虑 $m=2$ 和 $n=3$ 的情况，而且假设我们的敌手是不受限制的。如果头一个比较是 $A_1:B_1$，则这个敌手可以采取策略 A′(1,1)，此时需要 $.M.(0,1)+.M.(2,2)=3$ 个进一步的比较。如果第一个比较是 $A_1:B_3$，则这个敌手可以采取策略B(1,2)，此时需要 $.M\backslash(1,2)+/M.(1,2)=4$ 个进一步的比较。不管首先比较哪一对 $A_i:B_j$，这个敌手总可以保证至少必须进行 3 次进一步的比较，因此 $.M.(2,3)=4$。

用手来进行这些计算是不容易的，但是一台计算机却可以相当快地做出 $\lambda M\rho$

函数表。它有某些明显的对称性,例如

$$/M.(m,n) = .M\backslash(m,n) = \backslash M.(n,m) = .M/(n,m) \qquad (7)$$

借助于它,我们可以把 9 个函数减少到只有 4 个:$M.(m,n)$,$/M.(m,n)$,$/M\backslash(m,n)$以及$/M/(m,n)$。表 1 指出对于所有 $m,n\leqslant 10$ 所得到的值;我们的合并的敌手已经以这样的方式定义,即

$$.M.(m,n) \leqslant M(m,n) \quad \text{对于所有 } m,n \geqslant 0 \qquad (8)$$

这个关系包括定理 M 作为一个特殊情况,因为当 $|m-n|\leqslant 1$ 时我们的敌手将使用该定理的简单策略。

现在考虑为 M 函数所满足的某些简单关系:

$$M(m,n) = M(n,m) \qquad (9)$$

$$M(m,n) \leqslant M(m,n+1) \qquad (10)$$

表 1 由"敌手"得到的对于合并的下限

m \ n	$.M.(m,n)$ 1	2	3	4	5	6	7	8	9	10	$/M.(m,n)$ 1	2	3	4	5	6	7	8	9	10	m
1	1	2	2	3	3	3	3	4	4	4	1	2	2	3	3	3	3	4	4	4	1
2	2	3	4	5	5	6	6	6	7	7	1	3	4	4	5	5	6	6	7	7	2
3	2	4	5	6	7	7	8	8	9	9	1	3	5	6	7	7	8	8	9	9	3
4	3	5	6	7	8	9	10	10	11	11	1	4	5	7	8	9	9	10	10	11	4
5	3	5	7	8	9	10	11	12	12	13	1	4	6	8	9	10	11	12	12	13	5
6	3	6	7	9	10	11	12	13	14	15	1	4	6	8	10	11	12	13	14	14	6
7	3	6	8	10	11	12	13	14	15	16	1	4	7	9	10	12	13	14	15	16	7
8	4	6	8	10	12	13	14	15	16	17	1	5	7	9	11	13	14	15	16	17	8
9	4	7	9	11	12	14	15	16	17	18	1	5	8	10	11	13	15	16	17	18	9
10	4	7	9	11	13	15	16	17	18	19	1	5	8	10	12	14	15	17	18	19	10

m \ n	$/M\backslash(m,n)$ 1	2	3	4	5	6	7	8	9	10	$/M/(m,n)$ 1	2	3	4	5	6	7	8	9	10	m
1	$-\infty$	2	2	3	3	3	3	4	4	4	1	1	1	1	1	1	1	1	1	1	1
2	$-\infty$	2	4	4	5	5	6	6	7	7	1	3	3	4	4	4	4	5	5	5	2
3	$-\infty$	2	4	6	6	7	8	8	8	9	1	3	5	5	6	6	7	7	8	8	3
4	$-\infty$	2	5	6	8	8	9	10	10	11	1	4	5	7	7	8	9	9	9	10	4
5	$-\infty$	2	5	7	8	10	10	11	12	13	1	4	6	7	9	9	10	11	11	12	5
6	$-\infty$	2	5	7	9	10	12	13	14	14	1	4	6	8	9	11	11	12	13	14	6
7	$-\infty$	2	5	8	10	11	12	14	15	16	1	4	7	9	10	11	13	14	15	15	7
8	$-\infty$	2	6	8	10	12	13	15	16	17	1	5	7	9	11	12	14	15	16	17	8
9	$-\infty$	2	6	9	10	12	14	16	17	18	1	5	8	9	11	13	15	16	17	18	9
10	$-\infty$	2	6	9	11	13	15	16	18	19	1	5	8	10	12	14	15	17	18	19	10

$$M(k+m,n)\leqslant M(k,n)+M(m,n) \tag{11}$$

$$M(m,n)\leqslant \max(M(m,n-1)+1,\ M(m-1,n)+1) \text{ 对于 } m\geqslant 1, n\geqslant 1 \tag{12}$$

$$M(m,n)\leqslant \max(M(m,n-2)+1,\ M(m-1,n)+2 \text{ 对于 } m\geqslant 1, n\geqslant 2 \tag{13}$$

如果首先比较 $A_1:B_1$,由通常的合并过程即得出关系(12)。通过首先比较 $A_1:B_2$,即可类似地导出关系(13);如果 $A_1>B_2$,则需要 $M(m,n-2)$次进一步的比较,但如果 $A_1<B_2$,则我们可以把 A_1 插入到它适当的位置,并把$\{A_2,\cdots,A_m\}$同$\{B_1,\cdots,B_n\}$合并。推而广之,通过首先比较 $A_1:B_k$,而且如果 $A_1<B_k$ 便使用二叉查找,我们可以看出如果 $m\geqslant 1$ 和 $n\geqslant k$,我们有

$$M(m,n)\leqslant \max(M(m,n-k)+1, M(m-1,n)+1+\lceil \lg k\rceil) \tag{14}$$

结果是,对于所有 $m,n\leqslant 10$,$M(m,n)=.M.(m,n)$,所以表 1 实际上给出了合并的最优值。这可以通过使用(9)~(14)以及在习题 8、9 和 10 中给出的$(m,n)=(2,8)$,$(3,6)$和$(5,9)$的特殊构造得到证明。

另一方面,我们的敌手并不总是给出最好可能的下限;最简单的例子是 $m=3$,$n=11$,这时$.M.(3,11)=9$,但是 $M(3,11)=10$。为了看出在此种情况下这个"敌人"在哪儿"出错",我们必须研究他做此判断的理由;进一步检查揭示出,如果$(i,j)\neq(2,6)$,则这个敌手可以找到一个要求 10 次比较的策略;但当$(i,j)=(2,6)$时,没有策略胜过策略 A(2,4),使下限成为 $1+.M.(2,3)+.M.(1,8)=9$。通过把$\{A_1,A_2\}$同$\{B_1,B_2,B_3\}$以及$\{A_3\}$同$\{B_4,\cdots,B_{11}\}$合并来完成整个合并是必要的但不是充分的,所以在这种情况下下限并不是最佳的。

类似地,可以证明$.M.(2,38)=10$,而 $M(2,38)=11$,所以我们的敌手并没有好到足以解决 $m=2$ 的情况。但是有一类无穷多个值,对于这类值来说它是杰出的:

定理 K $M(m,m+2)=2m+1$ 对于 $m\geqslant 2$

$M(m,m+3)=2m+2$ 对于 $m\geqslant 4$

$M(m,m+4)=2m+3$ 对于 $m\geqslant 6$

证明 事实上可以以$.M.$代替 M 来证明这个结果;对于小的 m,这些结果已经通过计算机得到,所以可以假定 m 是足够大的。也可以假定,第一个比较是 $A_i:B_j$,其中 $i\leqslant\lceil m/2\rceil$。如果 $j\leqslant i$ 则使用策略 A′(i,i),对 d 用归纳法,$d\leqslant 4$,得到

$$.M.(m,m+d)\geqslant 1+.M.(i-1,i)+.M.(m+1-i,m+d-i)=$$
$$2m+d-1$$

如果 $j>i$,则使用策略 A$(i,i+1)$,对 m 用归纳法,得到

$$.M.(m,m+d)\geqslant 1+.M.(i,i)+.M.(m-i,m+d-i)=$$
$$2m+d-1 \quad \blacksquare$$

定理 K 的前两部分由 F. Hwang 和 S. Lin 于 1969 年得到。Paul Stockmeyer 和 Frances Yao 许多年后证明，在这 3 个公式中显示的模式一般都成立，即由以上的策略所导出的下限足以得到值 $M(m,m+d)=2m+d-1$，其中 $m\geqslant 2d-2$ [*SICOMP* **9** (1980)，85～90]。

上限　现在来考虑 $M(m,n)$ 的上限；好的上限对应于有效的合并算法。

当 $m=1$ 时，合并问题等价于一个插入问题，而且 $B_1,\cdots,B_n$ 之中有 $n+1$ 个位置是 A_1 可以插入进去的。对于这种情况，容易看出，具有 $n+1$ 个外部节点的任何一株扩展的二叉树均对应于某个合并方法的树（见习题 2）。因此，可以选择一株最优二叉树，它实现信息论的下限

$$1+\lfloor \lg n\rfloor = M(1,n) = \lceil \lg(n+1)\rceil \tag{15}$$

当然，二叉查找（6.2.1 小节）是达到这个值的一个简单方法。

$m=2$ 的情况是极为有趣的，但也困难得多，不过它已经被 R. L. Graham，F. K. Hwang 以及 S. Lin 完全解决了（见习题 11，12，13）；他们证明了一般的公式

$$M(2,n) = \left\lceil \lg \frac{7}{12}(n+1)\right\rceil + \left\lceil \lg \frac{14}{17}(n+1)\right\rceil \tag{16}$$

我们已经知道，当 $m=n$ 时，通常的合并过程是最优的，而当 $m=1$ 时，颇为不同的二叉查找过程是最优的。我们需要的是居中的方法，它把通常的合并算法同二叉查找组合在一起，并且保留两者最好的特性。公式(14)提出了下列合并算法，它是由 F. K. Hwang 和 S. Lin[*SICOMP* **1**(1972)，31～39]给出的：

算法 H（二叉合并）

H1. 如果 m 或 n 为 0，则停止。如果 $m>n$，则置 $t\leftarrow\lfloor \lg(m/n)\rfloor$ 并转向步骤 H4。否则置 $t\leftarrow\lfloor \lg(n/m)\rfloor$。

H2. 比较 $A_m:B_{n+1-2^t}$。如果 A_m 是较小的，则置 $n\leftarrow n-2^t$ 并返回步骤 H1。

H3. 利用二叉查找（它恰恰要求 t 次进一步的比较），把 A_m 插入到它在 $\{B_{n+1-2^t},\cdots,B_n\}$ 中的应有位置。如果 k 是使得 $B_k<A_m$ 的极大值，则置 $m\leftarrow m-1$ 和 $n\leftarrow k$。返回 H1。

H4.（步骤 H4 和 H5 同 H2 和 H3 类似，只是交换了 m 和 n，A 和 B 的作用）如果 $B_n<A_{m+1-2^t}$，则置 $m\leftarrow m-2^t$ 并返回步骤 H1。

H5. 插入 B_n 到它在诸 A 中的应有位置。如果 k 是使得 $A_k<B_n$ 的极大值，则置 $m\leftarrow k$ 和 $n\leftarrow n-1$，返回 H1。▌

作为这个算法的一个例子，表 2 示出 3 个键码 {087，503，512} 同 13 个键码 {061，154，⋯，908} 合并的过程；在这个例子中，需要进行 8 次比较。在每步当中被比较的元素以黑体示出。

表 2 二叉合并的例子

A			B													输出		
087	503	**512**	061	154	170	275	426	509	612	653	677	**703**	765	897	908			
087	503	**512**	061	154	170	275	426	509	612	**653**	677				703	765	897	908
087	503	**512**	061	154	170	275	426	509	612				653	677	703	765	897	908
087	503	**512**	061	154	170	275	426	509	**612**				653	677	703	765	897	908
087	**503**		061	154	170	275	**426**	509			512	612	653	677	703	765	897	908
087	**503**		061	154	170	275	426	**509**			512	612	653	677	703	765	897	908
087			061	**154**	170	275	426		503	509	512	612	653	677	703	765	897	908
087			**061**		154	170	275	426	503	509	512	612	653	677	703	765	897	908
			061	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908

设 $H(m,n)$是 Hwang 和 Lin 算法所要求的极大比较数。为了计算 $H(m,n)$，可以假定在步骤 H 3 中 $k=n$ 且在步骤 H5 中 $k=m$，因为通过对 m 用归纳法，我们将证明对所有的 $n\geqslant m-1$，$H(m-1,n)\leqslant H(m-1,n+1)$。于是当 $m\leqslant n$ 时，对于 $2^t m\leqslant n<2^{t+1}m$，有

$$H(m,n) = \max(M(m,n-2^t)+1, H(m-1,n)+t+1) \tag{17}$$

以 $2n+\epsilon$ 代替 n，$\epsilon=0$ 或 1，得到，对于 $2^t m\leqslant n<2^{t+1}m$，

$$H(m,2n+\epsilon) = \max[H(m,2n+\epsilon-2^{t+1})+1, H(m-1,2n+\epsilon)+t+2]$$

对 n 用归纳法，得出

$$H(m,2n+\epsilon) = H(m,n)+m \quad 对于\ m\leqslant n, \epsilon=0\ 或\ 1 \tag{18}$$

容易看出，当 $m\leqslant n<2m$ 时，$H(m,n)=m+n-1$；因此重复地应用(18)即得一般的公式

$$H(m,n) = m+\lfloor n/2^t\rfloor-1+tm \quad 对于\ m\leqslant n, t=\lfloor\lg(n/m)\rfloor \tag{19}$$

这意味着，对所有的 $n\geqslant m$ 有 $H(m,n)\leqslant H(m,n+1)$，证实了我们关于 H3 这一步的归纳假设。

置 $m=\alpha n$ 和 $\theta=\lg(n/m)-t$，当 $n\to\infty$ 时，给出

$$H(\alpha n,n) = \alpha n(1+2^\theta-\theta-\lg\alpha)+O(1) \tag{20}$$

由等式 5.3.1-(36)我们知道，$1.9139<1+2^\theta-\theta\leqslant 2$；因此，(20)可以同信息论的下限(3)相比较，Hwang 和 Lin 已经证明(见习题 17)

$$H(m,n) < \left\lceil \lg\binom{m+n}{m}\right\rceil + \min(m,n) \tag{21}$$

Hwang-Lin 的二叉合并算法并不总是给出最优的结果，但它有很大的长处，即它编写程序相当容易。当 $m=1$ 时，该算法归结为“非中心的二叉查找”，当 $m\approx n$ 时，它归结为通常的合并过程，所以它是这两个方法之间的一种卓越的折衷。而且，在许多情况下它是最优的(见习题 16)。F. K. Hwang 和 D. N. Deutsch[*JACM* **20**(1973)，148～159]，G. K. Manacher，[*JACM* **26**(1979)，434～440]，以及最突出的是 C. Christen[*FOCS* **19**(1978)，259～266]，已经找到改进的算法。Christen 的合并过程，称为向前测试-向后插入，当 $n/m\to\infty$ 时比算法 H 节省大约 $m/3$ 的比较，而且当 $5m-3\leqslant n\leqslant 7m+2[m\ 偶]$时，Christen 过程达到下限 $M.(m,n)=\lfloor(11m+n-3)/$

4 」。因此，在这样的情况下（以及值得注意地，在我们的对手下限情况下也是），它是最优的。

公式(18)提示，M 函数本身可以满足

$$M(m,n) \leqslant M(m, \lfloor n/2 \rfloor) + m \tag{22}$$

这实际上是真的(见习题 19)。$M(m,n)$的表提示了其它若干似乎正确的关系，诸如

$$M(m+1,n) \geqslant 1 + M(m,n) \geqslant M(m,n+1) \quad 对于\ m \leqslant n \tag{23}$$

$$M(m+1,n+1) \geqslant 2 + M(m,n) \tag{24}$$

但这些不等式尚未得到证明。

习　题

1.[*15*] 试求 $M(m,n)$和 5.3.1 小节中定义的函数 S 之间有趣的关系[提示：考虑 $S(m+n)$]。

▶2.[*22*] 当 $m=1$ 时，没有多余比较的每个合并算法都定义了一株具有 $\binom{m+n}{m}=n+1$ 个外部节点的扩展二叉树。试证明：反过来，每一株具有 $n+1$ 个外部节点的扩展二叉树都对应于 $m=1$ 的合并算法。

3.[*M24*] 证明对所有 n，$.M.(1,n)=M(1,n)$。

4.[*M42*] 对所有 m 和 n，$.M.(m,n) \geqslant \left\lceil \lg \binom{m+n}{m} \right\rceil$吗？

5.[*M30*] 证明 $.M.(m,n) \leqslant .M \backslash (m,n+1)$。

6.[*M26*] 定理 K 中所述的证明要求计算机验证大量不同的情形，如何能大大地减少这种验证的数目？

7.[*21*] 证明(11)。

▶8.[*24*] 通过找出一个算法，它使用至多 6 次比较把 2 个元素同其他 8 个元素合并起来，来证明 M(2,8)≤6。

9.[*27*] 证明至多用 7 步便可以把 3 个元素同 6 个元素合并起来。

10.[*33*] 证明 5 个元素可以在至多 12 步中同 9 个元素合并起来[提示：根据敌手的经验提议先比较 $A_1:B_2$，然后如果 $A_1<B_2$，则比较 $A_5:A_8$]。

11.[*M40*](F. Hwang S. Lin) 对于 $k \geqslant 0$，令 $g_{2k}=\left\lfloor 2^k \frac{17}{14} \right\rfloor$，$g_{2k+1}=\left\lfloor 2^k \frac{12}{7} \right\rfloor$，使得$(g_0,g_1,g_2,\cdots)=(1,1,2,3,4,6,9,13,19,27,38,54,77,\cdots)$。证明，在最坏的情况下，它花费 t 次以上的比较把 2 个元素同 g_t 个元素合并；但是 2 个元素可以在至多 t 步中同 g_t-1 个元素合并[提示：证明，如果 $n=g_t$ 或 $n=g_t-1$，且如果要在 t 次比较中合并$\{A_1,A_2\}$和$\{B_1,B_2,\cdots,B_n\}$，则第一步以比较 $A_2:B_{g_{t-1}}$为最好]。

12.[*M21*] 设 $R_n(i,j)$是为对不同的对象$\{\alpha,\beta,X_1,X_2,\cdots,X_n\}$排序所需要的最少比较数，给定关系

$$\alpha < \beta, \quad X_1 < X_2 < \cdots < X_n, \quad \alpha < X_{i+1}, \quad \beta > X_{n-j}$$

（当 $i \geqslant n$ 或 $j \geqslant n$ 时，条件 $\alpha < X_{i+1}$ 或 $\beta > X_{n-j}$ 为空，因此 $R_n(n,n) = M(2,n)$）。

显然，$R_n(0,0)=0$。证明：对于 $0 \leqslant i \leqslant n, 0 \leqslant j \leqslant n, i+j>0$，

$$R_n(i,j) = 1 + \min\Big(\min_{1\leqslant k\leqslant i} \max(R_n(k-1,j), R_{n-k}(i-k,j)),$$
$$\min_{1\leqslant k\leqslant j} \max(R_n(i,k-1), R_{n-k}(i,j-k))\Big)$$

13. [*M42*] (R. L. Graham)说明习题 12 中递推关系的解可以表达如下，根据规则

$$G(x) = \begin{cases} 1 & \text{如果 } 0 < x \leqslant \frac{5}{7} \\ \frac{1}{2} + \frac{1}{8}G(8x-5) & \text{如果 } \frac{5}{7} < x \leqslant \frac{3}{4} \\ \frac{1}{2}G(2x-1) & \text{如果 } \frac{3}{4} < x \leqslant 1 \\ 0 & \text{如果 } 1 < x < \infty \end{cases}$$

对于 $0 < x < \infty$，定义函数 $G(x)$。

见图 38，由于 $R_n(i,j) = R_n(j,i)$ 以及由于 $R_n(0,j) = M(1,j)$，可以假定 $1 \leqslant i \leqslant j \leqslant n$。令 $p = \lfloor \lg i \rfloor, q = \lfloor \lg j \rfloor, r = \lfloor \lg n \rfloor$，并令 $t = n - 2^r + 1$。于是

$$R_n(i,j) = p + q + S_n(i,j) + T_n(i,j)$$

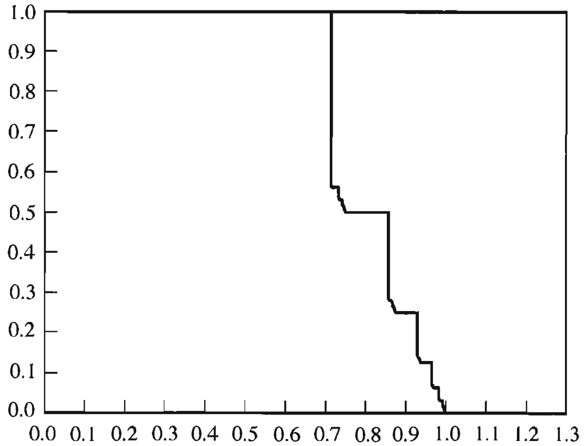


图 38 Graham 的函数(见习题 13)

其中 S_n 和 T_n 为 0 或 1 的函数：

$$S_n(i,j) = 1 \quad \text{当且仅当} \quad q < r \text{ 或}(i - 2^p \geqslant u \text{ 且 } j - 2^r \geqslant u)$$

$$T_n(i,j) = 1 \quad \text{当且仅当} \quad p < r \text{ 或}\left(t > \frac{6}{7}2^{r-2} \text{ 且 } i - 2^r \geqslant v\right)$$

其中 $u = 2^p G(t/2^p)$ 且 $v = 2^{r-2}G(t/2^{r-2})$。

这可能是需要解决的最麻烦的递推关系！

14. [*41*] (F. K. Hwang) 对于 $k \geqslant 3$ 设 $h_{3k} = \lfloor \frac{43}{28}2^k \rfloor - 1$, $h_{3k+1} = h_{3k} + 3 \cdot 2^{k-3}$, $h_{3k+2} = \lfloor \frac{17}{7}2^k - \frac{6}{7} \rfloor$，且设初始值被定义成使得 $(h_0, h_1, h_2, \cdots) = (1,1,2,2,3,4,5,7,9,11,14,18,23,29,$

38,48,60,76…)。证明对所有 t,$M(3,h_t)>t$ 和 $M(3,h_t-1)\leqslant t$,由此确定对于所有 n,$M(3,n)$ 的准确值。

15.[*12*] 二叉合并算法的步骤 H1 可能要求对于 $n\geqslant m$,计算$\lfloor \lg(n/m)\rfloor$。说明要如何做,才能避免使用除法或对数,来方便地计算它。

16.[*18*] 当 $1\leqslant m\leqslant n\leqslant 10$ 时,对于什么样的 m 和 n,Hwang 和 Lin 的二叉合并算法是最优的?

17.[*M25*] 证明(21)[提示:这个不等式不是非常严格的]。

18.[*M40*] 分析二叉合并所使用的平均比较次数。

▶19.[*23*] 证明 M 函数满足(22)。

20.[*20*] 证明如果对所有 $m\leqslant n$,$M(m,n+1)\leqslant M(m+1,n)$,则对所有 $m\leqslant n$,$M(m,n+1)\leqslant 1+M(m,n)$。

21.[*M47*] 证明或否定(23),(24)。

22.[*M43*] 分析为合并 m 个事物和 n 个事物所需要的极小平均比较次数。

23.[*M31*] (E. Reingold) 设$\{A_1,A_2,\cdots,A_n\}$和$\{B_1,B_2,\cdots,B_n\}$是各包含 n 个元素的集合。试考虑一个算法,它试图仅通过比较元素的相等性,来测试这两个集合的相等性。于是,此算法提出了一个问题,即:对某个 i 和 j,"$A_i=B_j$ 吗?",根据答案的不同,此算法将产生不同的分支。

试通过定义一适当的敌手,证明任何这样的算法,在它最坏的情况下至少必须进行$\frac{1}{2}n(n+1)$次比较。

24.[*22*] (E. L. Lawler) 把 m 个元素同 $n\geqslant m$ 个元素合并的下列算法,所需要的极大比较次数是多少?"置 $t\leftarrow\lfloor \lg(n/m)\rfloor$,并使用算法 5.2.4M 合并 $A_1,A_2,\cdots,A_m$ 和 $B_{2^t},B_{2\cdot 2^t},\cdots,B_{q\cdot 2^t}$,其中 $q=\lfloor n/2^t\rfloor$。然后把每个 A_j 插入它在 B_k 当中的正确位置。"

▶25.[*25*] 假设(x_{ij})是具有非递减的行和列的一个 $m\times n$ 矩阵:对于 $1\leqslant i<m$,$x_{ij}\leqslant x_{(i+1)j}$;对于 $1\leqslant j<n$,$x_{ij}\leqslant x_{i(j+1)}$。证明,如果所有的比较都在 x 和某个矩阵元素之间进行,则 $M(m,n)$是为确定一个给定的数 x 是否在矩阵中出现,所需要的极小比较次数。

*5.3.3 极少比较选择

当我们寻找最好的过程以选择 n 个元素的第 t 个最大者时,出现了一类类似的有趣问题。

这个问题的历史可回溯到 C. L. Dodgson 牧师关于草地网球锦标赛的有趣的(尽管是严肃的)试验,它出现在 St. James,*Gazette* ,1883 年 8 月 1 日,5～6 上。Dodgson 当然比 Lewis Carroll 更为有名,他关心的是在网球锦标赛中过去(而且现在仍然如此)颁发奖金的不公平方式。例如,考虑图 39 中标号为 01,02,…,32 的 32 位选手之间进行的一场典型的"淘汰赛"。在决赛中,选手 01 击败了 05,所以显然选手 01 是冠军而且他应获头等奖。但在如下的问题上却出现了不公正,即选手 05 通常获得二等奖,尽管他可能不是第二好的选手。在这场锦标赛中,即使你比一半的选手都要差,却仍有可能获得二等奖。事实上,如同 Dodgson 所注意到的,第二个最好的选手当且仅当他在锦标赛开始时和冠军处于不同的两半中时才能获二等奖;若有 2^n 个选手,则这以 $2^{n-1}/(2^{n-1})$的概率出现,所以不是第二好的选手几乎有一半

的机会可获得二等奖！如果半决赛的失败者(图 39 中的 25 和 17 号选手)竞争三等奖，则第三个最好者接受三等奖的可能性更小。

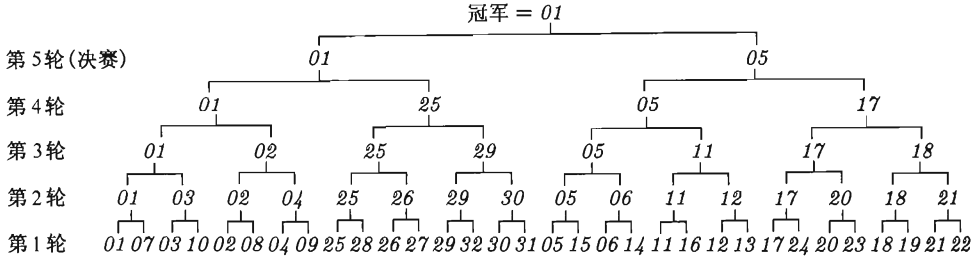


图 39 32 位选手的淘汰赛

因此，Dodgson 提出用传递性排次序的方法来设计一个锦标赛，以确定真正的第二和第三好的选手(换言之，如果选手 A 击败选手 B，而选手 B 击败选手 C，则假定 A 将击败 C)。他设计了一个过程，在这个过程中，允许失败者进一步比赛，直到确实知道他们劣于其他三个选手为止。图 40 所示为 Dodgson 方案的一个示例，它是有待同图 39 配合进行的补充比赛。该方案尽量把迄今具相等记录的选手配对比赛，并且避免两个已被同一个选手击败的选手进行比赛。例如，在第一轮中，16 败于 11 和 13 败于 12；在第二轮中 16 败于 13，由此就取消了 16，因为已经知道他低于 11，12 和 13 了。在第三轮中，不允许 19 同 21 进行比赛，因为他们俩都为 18 所击败，故我们不能自动地取消 19 对 21 的失败者。

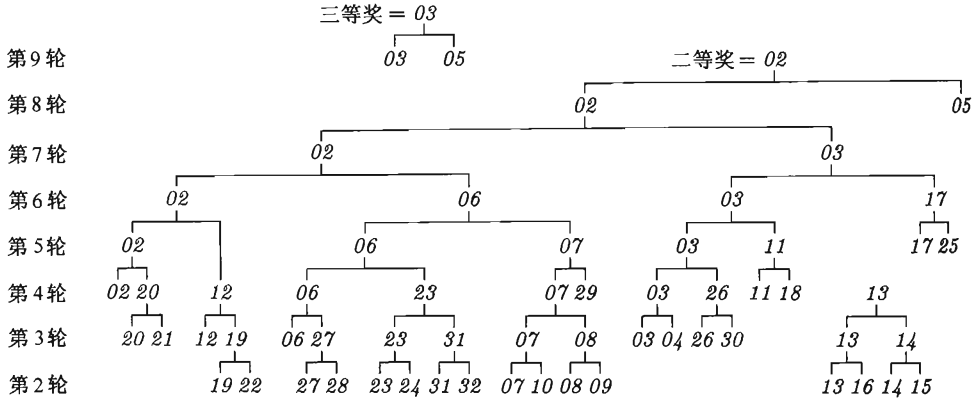


图 40 Lewis Carroll 的草地锦标赛(同图 39 合在一起进行)

假如我们能说 Lewis Caroll 的锦标赛确实是最优的，那就好了。但可惜情况并非如此。他在 1883 年 7 月 23 日的日记中指出，他用了大约 6 个小时来排方案，他觉得“现在我们的(网球)赛期已过去了这么多，因此比起写出来，倒不如立即就进行比赛更好些。”他的过程中要做比需要的还多的比较，而且该过程没有足够精确地描述出来，从而可以被称为一个算法。另一方面，从并行计算的观点看，它有某些颇为

有趣的方面。而且对于一个网球锦标赛说来,它似乎是一个卓越的计划,因为他取得了某些戏剧性的效果;例如,他确定,两个决赛选手应该不参加第 5 轮比赛,而参加第 6 轮和第 7 轮中进行的附加比赛。但是比赛的组织者大概认为这个提案太逻辑化了,因此看来从未试用过 Carroll 的系统。而代之使用一个"种子选手"方法,来把最好的选手分在树的不同部分。

在 1929~1930 年的数学研讨会中,Hugo Steinhaus 提出了当共有 $n\geqslant 2$ 个选手时,为确定在一次锦标赛中的第一和第二最好的选手,所需要的网球比赛的极小场次问题。J. Schreier[*Mathesis Polska* **7**(1932),154~160]给出了至多需要 $n-2+\lceil \lg n\rceil$ 场比赛的一个过程,基本上使用了我们称做树选择排序方法中的前两个阶段同样的方法(参见 5.2.3 小节,图 23),避免涉及 $-\infty$ 的多余比较。Schreier 还宣称,$n-2+\lceil \lg n\rceil$ 是最好的,但他的证明如同 J. Slupecki 试图另给的证明[*Colloquium Mathematicum* **2**(1951),286~290]一样是不正确的。32 年之后,S. S. Kislitsyn 最后发表了一个正确的但相当复杂的证明[*Sibirskiy Mat. Zhurnal* ,**5**(1964),557~564]。

令 $V_t(n)$ 表示为确定 n 个元素的第 t 个最大者所需要的极小比较次数,$1\leqslant t\leqslant n$,并令 $W_t(n)$ 是为了确定最大者,第二个最大者,…,以及第 t 个最大者等全部 t 个元素所需要的极小比较次数。由对称性,我们有

$$V_t(n) = V_{n+1-t}(n) \tag{1}$$

而且显然有

$$V_1(n) = W_1(n) \tag{2}$$

$$V_t(n) \leqslant W_t(n) \tag{3}$$

$$W_n(n) = W_{n-1}(n) = S(n) \tag{4}$$

在引理 5.2.3M 中我们已经发现

$$V_1(n) = n-1 \tag{5}$$

实际上,关于这个事实,有一个惊人地简单证明,因为在一次锦标赛中,除了冠军外,每个选手都至少输了一场! 通过扩展这一思想,并使用一个"敌手",就可以不太困难地证明 Schreier-Kislitsyn 定理。

定理 S　当 $n\geqslant 2$ 时,$V_2(n)=W_2(n)=n-2+\lceil \lg n\rceil$。

证明　假设有 n 个选手参加一次锦标赛。这个锦标赛通过某种给定的过程确定了第二个最好的选手,并令 a_j 是输了 j 场或更多场比赛的选手数。于是,比赛的总场数是 $a_1+a_2+a_3+\cdots$。如果不同时确定出冠军来的话,我们不能确定第二个最好的选手(参见习题 2),所以用以前的论证说明 $a_1=n-1$。为完成这个证明,将证明定有某个比赛结果的序列,使得 $a_2\geqslant\lceil \lg n\rceil-1$。

假设在锦标赛结束时,冠军已经迎战(并击败了)p 个选手;这些人当中有一个

是第二个最好者,那么这些人中的其他人就另外至少输了一场,所以 $a_2 \geqslant p-1$。因此可以通过这样一个方法来完成这个证明,即通过构造一个敌手,该敌手以这样一个办法来制定比赛的结果,即冠军至少必须同$\lceil \lg N \rceil$个其他人较量。

设此敌手在下列情况下断言 A 比 B 更好,即:如果 A 以前未败过,而 B 至少输了一次,或者如果两者都未败过而此时 B 比 A 赢得少些。在其它情况下,敌手可以做出同某个偏序相一致的任何判断。

考虑整个锦标赛的结果,这次锦标赛的角逐已经为这样一个敌手所决断。即当且仅当 $A=B$ 或 A 强过第一个击败 B 的选手时我们说"A 强于 B"(按这个说法,对每个选手来说,只有第一次败绩是关键的,失败者随后的比赛则被忽略。按照敌手的规则,任何首先击败另一个人的人都必须以前未败过)。由此得出,赢得头 p 场的人,在这 p 场比赛的基础上至多强过 2^p 个选手(这对于 $p=0$ 是显然的,而对于 $p>0$,第 p 次比赛是迎战以前败过了的或至多强过 2^{p-1}个选手的某个人)。冠军强于每个人,所以他必须至少赢得了$\lceil \lg n \rceil$场比赛。 ▎

在极小化极大的意义下,定理 S 完全解决了求第二个最好选手的问题。事实上,习题 6 说明,当预先已知诸元素的一个任意偏序时,对于求出一个集合的第二个最大的元素所需要的极小比较次数,有可能给出一个简单的公式。

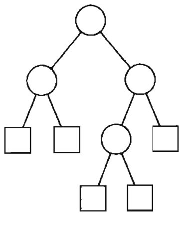
如果 $t>2$ 如何? 在上边引证的文章中,Kislitsyn 考虑过 t 的更大的值并证明

$$W_t(n) \leqslant n-t+\sum_{n+1-t<j\leqslant n} \lceil \lg j \rceil \qquad \text{对于 } n \geqslant t \tag{6}$$

对于 $t=1$ 和 $t=2$,已经看到,在这个公式中等式实际上成立;对于 $t=3$,它能稍加改进(见习题 21)。

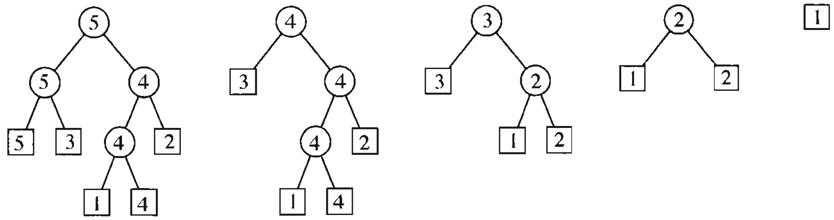
我们将说明树选择的前 t 个阶段至多需要 $n-t+\Sigma_{n+1-t<j\leqslant n}\lceil \lg j \rceil$次比较,并忽略涉及 $-\infty$ 的所有比较,以证明 Kislitsyn 定理。有意思的是,由等式 5.3.1-(3),当 $t=n$ 以及当 $t=n-1$ 时,(6)的右边等于 $B(n)$;因此,树选择和二元插入在排序下给出同样的上限,尽管它们是十分不同的方法。

令 α 是具有 n 个外部节点的扩展二叉树,并令 π 是$\{1,2,\cdots,n\}$的一个排列。以对称的次序从左到右地把 π 的元素放进外部节点,并像在树选择中那样按淘汰锦标赛的规则填入内部节点。当得到的树被施以重复的选择操作时,它就定义了一个序列 $c_{n-1}, c_{n-2}, \cdots, c_1$,其中 c_j 是当元素 $j+1$ 已被 $-\infty$ 代替时,为把元素 j 引到树的根部所需要的比较次数。例如,如果 α 是树



(7)

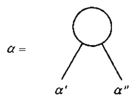
而且如果 $\pi=5\ 3\ 1\ 4\ 2$,则逐次得到如下的树



$c_4=1$　　　　$c_3=2$　　　　$c_2=0$　　　　$c_1=0$

如果 π 换成 3 1 5 4 2,则序列 $c_4c_3c_2c_1$ 将是 2110。容易看出 c_1 总是零。

令 $\mu(\alpha,\pi)$是由 α 和 π 确定的多重集合$\{c_{n-1},c_{n-2},\cdots,c_1\}$。如果



而且如果元素 1 和 2 不同时在 α'中出现,也不同时在 α''中出现,则容易看出对于适当的排列 π'和 π''

$$\mu(\alpha,\pi)=(\mu(\alpha',\pi')+1)\uplus(\mu(\alpha'',\pi'')+1)\uplus\{0\} \tag{8}$$

其中$(\mu+1)$表示对 μ 的每个元素加 1 得到的多重集合(参见习题 7)。另一方面,如果元素 1 和元素 2 都出现于 α'中,则有

$$\mu(\alpha,\pi)=(\mu(\alpha',\pi')+\varepsilon)\uplus(\mu(\alpha'',\pi'')+1)\uplus\{0\}$$

其中 $\mu+\varepsilon$ 表示把 1 加到 μ 的某些元素而把 0 加到其它元素得到的一个多重集合。当 1 和 2 都出现于 α''中时,类似的公式成立。我们说多重集合 μ_1 高于 μ_2,如果 μ_1 和 μ_2 两者有相同的元素个数,而且对于所有的 k,μ_1 的第 k 个最大的元素大于或等于μ_2 的第 k 个最大的元素;而且,在下述意义下,对于所有的排列 π,定义 $\mu(\alpha)$为最高的 $\mu(\alpha,\pi)$:对所有的 π,$\mu(\alpha)$高于 $\mu(\alpha,\pi)$,而且对于某个 π,$\mu(\alpha)=\mu(\alpha,\pi)$。上边的公式表明

$$\mu(\Box)=\phi,\ \mu\left(\begin{smallmatrix} & \bigcirc & \\ \swarrow & & \searrow \\ \alpha' & & \alpha''\end{smallmatrix}\right)=(\mu(\alpha')+1)\uplus(\mu(\alpha'')+1)\uplus\{0\} \tag{9}$$

因此,*$\mu(\alpha)$是从 α 的根到它的内部节点的所有距离的多重集合*。

只要读者遵循上面的思路,现在就能看出,我们已为 Kislitsyn 定理(6)的证明做好了准备;确实 $W_t(n)$小于或等于 $n-1$ 加 $\mu(\alpha)$的 $t-1$ 个最大元素,其中 α 是在树选择排序中使用的任何树,当

$$\begin{aligned}\mu(\alpha)&=\{\lfloor\lg 1\rfloor,\lfloor\lg 2\rfloor,\cdots,\lfloor\lg(n-1)\rfloor\}\\&=\{\lceil\lg 2\rceil-1,\lceil\lg 3\rceil-1,\cdots,\lceil\lg n\rceil-1\}\end{aligned} \tag{10}$$

时,可以取 α 为具有 n 个外部节点的完备二叉树(参见 2.3.4.5 小节)。当考虑这个多重集合的 $t-1$ 个最大的元素时,就得到公式(6)。

Kislitsyn 定理给出了对于 $W_t(n)$的一个好的上限;他注意到 $V_3(5)=6<W_3(5)=7$,但是他未能得到比起对于 $W_t(n)$来对于 $V_t(n)$的一个更好的上限。A. Hadian 和 M. Sobel 发现了使用替代选择,而不用树选择(见 5.4.1 小节),来解决这个问题的一个办法。他们的公式[Univ. of Minnesota, Dept. of Statistics Report **121** (1969)]

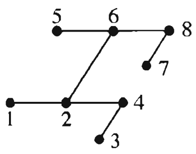
$$V_t(n) \leqslant n-t+(t-1)\lceil \lg(n+2-t) \rceil \qquad n \geqslant t \tag{11}$$

与(6)中 $W_t(n)$的 Kislitsyn 的上限类似,只是在(6)的和中的每项都已换成最小的项。

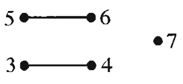
通过使用下列构造,即可证明 Hadion 和 Sobel 定理(11):首先建立对于 $n-t+2$ 个项目的一次淘汰锦标赛的二叉树(这花费 $n-t+1$ 次比较)。最大的项目大于 $n-t+1$ 个其它项目,所以它不可能是第 t 个最大的。不管它出现在树的哪一个外部节点处,都以保留的 $t-2$ 个元素之一代替它,在由此所得的 $n-t+2$ 个元素中,找出其最大的元素;这至多要求$\lceil \lg(n+2-t) \rceil$次比较,因为仅仅需要重新计算树中的一条路径。对于保留的每个元素重复这个操作,总共重复 $t-2$ 次。最后,以 $-\infty$ 代替当前最大的元素,并确定剩下的 $n+1-t$ 个元素中的最大者;这至多要求$\lceil \lg(n+2-t) \rceil-1$ 次比较,而且它把原来集合的第 t 个最大元素引向树的根。把这些比较加起来就得到(11)。

在关系(11)中,每当 $n+1-t$ 给出更好的值时(如同 $n=6, t=3$ 时那样),我们在式子右边当然将以 $n+1-t$ 来代替 t。奇怪的是,这个公式对于 $V_7(13)$给出了比对于 $V_6(13)$更小的限。对于 $n \leqslant 6$,(11)中的上限是精确的,但当 n 和 t 变大时,有可能得到对于 $V_t(n)$的好得多的估计。

例如,下列漂亮的方法(由 David G. Doren 给出)可以用来说明 $V_4(8) \leqslant 12$。设元素为 $X_1, \cdots, X_8$;首先比较 $X_1:X_2$ 和 $X_3:X_4$ 以及两个胜利者,并对 $X_5:X_6$ 和 $X_7:X_8$ 以及它们的胜利者也同样进行比较。重新标出这些元素,使得 $X_1<X_2<X_4>X_3, X_5<X_6<X_8>X_7$,然后比较 $X_2:X_6$;由对称性,假定 $X_2<X_6$,于是我们有配置



(现在 X_1 和 X_8 已脱离比赛,我们必须求$\{X_2, \cdots, X_7\}$的第三个最大者)。比较 $X_2:X_7$ 并丢弃较小者;在最坏的情况下,我们有 $X_2<X_7$,而且必须求出



的第三个最大者。这可在另外 $V_3(5)-2=4$ 次比较中完成,因为实现 $V_3(5)=6$ 的(11)的过程是通过比较两个不相交的元素对开始的。

这种类型的其它技巧可用来产生表 1 所示的结果;尚未发明一般的方法。1996年,W. Gasarch, W. Kelly 和 W. Pugh[*SIGACT News* **27**, 2(June 1996), 88～96],利用一个计算机查找,证明对于 $V_4(9) = V_6(9)$ 和 $V_5(10) = V_6(10)$ 所列的值是最优的。

当 t 很小时,David G. Kirkpatrick 已经得到了对于选择问题的相当好的下界[*JACM* **28**(1981), 150～165],如果 $2 \leqslant t \leqslant (n+1)/2$,我们有

$$V_t(n) \geqslant n + t - 3 + \sum_{j=0}^{t-2} \left\lceil \lg \frac{n-t+2}{t+j} \right\rceil \tag{12}$$

在他的博士论文[U. of Toronto, 1974]中,Kirkpatrick 还证明了

$$V_3(n) \leqslant n + 1 + \left\lceil \lg \frac{n-1}{4} \right\rceil + \left\lceil \lg \frac{n-1}{5} \right\rceil \tag{13}$$

对于所有整数 n 的 $\lg \frac{5}{3} \approx 74\%$,这个上限匹配下限(12),而且它至多比(12)超过 1。Kirkpatrick 的分析使人很自然地去猜测,对于所有 $n > 4$,(13)中的等式成立,但是 Jutta Eusterbrock 发现令人惊讶的反例 $V_3(22) = 28$[*Discrete Applied Math.* **41**(1993), 131～137]。对于较大的 t 值,S. W. Bent 和 J. W. John 发现改进了的下限(参见习题 27):

$$V_t(n) \geqslant n + m - 2\lceil \sqrt{m} \rceil, \quad m = 2 + \left\lceil \lg\left(\binom{n}{t} \Big/ (n+1-t) \right) \right\rceil \tag{14}$$

这个公式特别证明了

$$V_{\alpha n}(n) \geqslant \left(1 + \alpha \lg \frac{1}{\alpha} + (1-\alpha) \lg \frac{1}{1-\alpha}\right) n + O(\sqrt{n}) \tag{15}$$

表 1 对于 $V_t(n)$ 已知的最好上界

n	$V_1(n)$	$V_2(n)$	$V_3(n)$	$V_4(n)$	$V_5(n)$	$V_6(n)$	$V_7(n)$	$V_8(n)$	$V_9(n)$	$V_{10}(n)$
1	0									
2	1	1								
3	2	3	2							
4	3	4	4	3						
5	4	6	6	6	4					
6	5	7	8	8	7	5				
7	6	8	10	10*	10	8	6			
8	7	9	11	12	12	11	9	7		
9	8	11	12	14	14*	14	12	11	8	
10	9	12	14*	15	16**	16**	15	14*	12	9

* 在这些情况下,习题 10-12 给出改进等式(11)的构造。

** 见 K. Noshita, *Trans. of the IECE of Japan*, E59, 12(DEC. 1976), 17～18

一个线性方法 当 n 是奇数且 $t=\lceil n/2\rceil$ 时，第 t 个最大的（以及第 t 个最小的）元素称做中值。按照(11)，我们可以在 $\approx\frac{1}{2}n\lg n$ 次比较中求出 n 个元素的中值；尽管只要求少得多的信息，但这大约只比排序快一倍。若干年来许多人一直致力于寻求当 t 和 n 很大时对于(11)的改进。最后，在 1971 年，Manuel Blum 发现了仅仅需要 $O(n\log\log n)$ 个步骤的一个方法。Blum 解决这个问题的途径，提示了一类新技术，它导致了 R. Rivest 和 R. Tarjan 给出的下列构造[*J. Comp. and Sys. Sci.* **7** (1973)，448～461]：

定理 L *当* $n>32$ *时，对于* $1\leqslant t\leqslant n$，$V_t(n)\leqslant 15n-163$。

证明 当 n 很小时，这定理是显然的，因为对于 $32<n\leqslant 2^{10}$ 有 $V_t(n)\leqslant S(n)\leqslant 10n\leqslant 15n-163$。通过加上至多 13 个虚构的“$-\infty$”元素，对于某个整数 $q\geqslant 73$，我们可以假定 $n=7(2q+1)$。下列方法可以用来选择第 t 个最大的元素：

步骤 1 把这些元素分成每组 7 个元素的 $2q+1$ 组，对每组排序。这至多花费 $13(2q+1)$ 次比较。

步骤 2 求在步骤 1 中得到的 $2q+1$ 个中值元素的中值，称它为 x。通过对 q 用归纳法可证，这至多花费 $V_{q+1}(2q+1)\leqslant 30q-148$ 次比较。

步骤 3 不同于 x 的 $n-1$ 个元素现在已经分成为 3 个集合（见图 41）：

$4q+3$ 个已知大于 x 的元素（区域 B）；

$4q+3$ 个已知小于 x 的元素（区域 C）；

$6q$ 个元素，它们同 x 的关系是未知的（区域 A、D）。

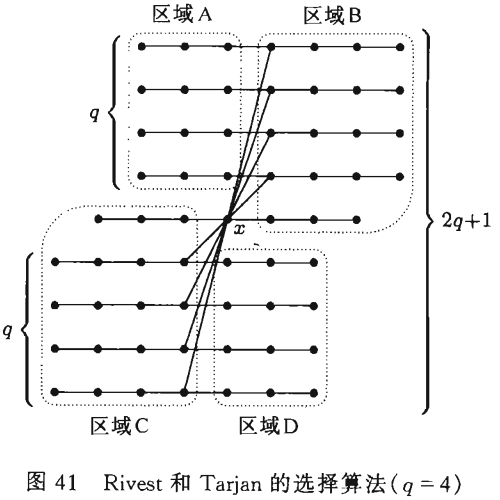


图 41 Rivest 和 Tarjan 的选择算法（$q=4$）

通过进行 $4q$ 次附加的比较，我们就能确切地说出区域 A 和 D 中哪些元素小于 x（首先把 x 和每个三元组的中间元素做比较）。

步骤 4 对某个 r，现在找到了 r 个大于 x 的元素和 $n-1-r$ 个小于 x 的元素。

如果 $t = r+1$，则 x 是答案；如果 $t < r+1$，则需要求 r 个大的元素中第 t 个最大的；如果 $t > r+1$，则需要求 $n-1-r$ 个小元素中的第 $(t-1-r)$ 个最大的。要点是 r 和 $n-1-r$ 两者都小于或者等于 $10q+3$（区域 A 和 D 加上 B 或 C 的大小）。因此，通过对 q 用归纳法可证，这个步骤至多需要 $15(10q+3)-163$ 次比较。

比较的次数至多为

$$13(2q+1)+30q-148+4q+15(10q+3)-163 = 15(14q-6)-163$$

由于我们是从至少 $14q-6$ 个元素开始的，证明就完成了。 ▌

定理 L 证明了选择总可在"线性时间"内完成，即证明了 $V_t(n)=O(n)$。当然，这个证明所使用的方法稍微有些粗糙，因为它丢弃了步骤 4 中好的信息。关于这一问题的研究已经得到更漂亮的上限；例如，A. Schönhage，M. Paterson 及 N. Pippenger[*J. Comp. Sys. Sci.* **13**(1976)，184～199]已经证明，为求平均值所需要的极大比较次数至多是 $3n+O(n \log n)^{3/4}$。关于一个下限和更新的结果的参考文献，参见习题 23。

平均数　除了把极大比较次数极小化外，我们还可以要求一个算法，对于随机

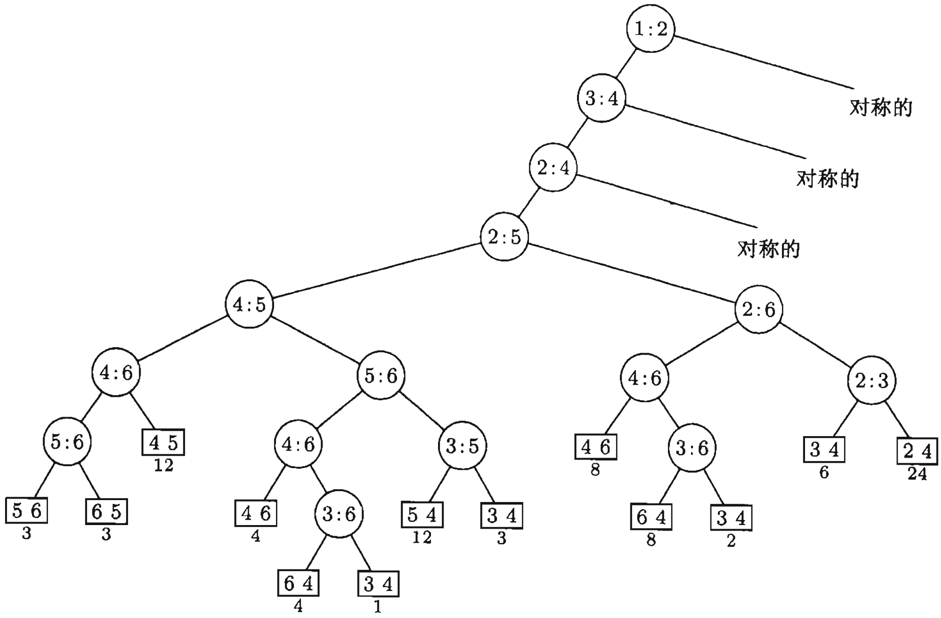


图 42　选择 $\{X_1, X_2, X_3, X_4, X_5, X_6\}$ 的第二个最大者的过程，平均使用 $6\frac{1}{2}$ 次比较。每个"对称的"分支和它的兄弟相同，只是名字按某种适当的方式排列过。当已知 X_j 是第二个最大者和 X_k 是最大者时，外部节点包含"$j\ k$"；写在每个外部节点下的数字表示通向这个节点要做多少次排列

次序下的对象,使得平均比较次数极小化。和通常一样,这个问题要比极小化极大值问题难得多;确实,甚至对于 $t=2$ 的情况极小化平均值问题也尚未解决。Claude Picard 在他的*Théorie des Questionnaires* (1965)一书中提到了这个问题,而 Milton Sobel 做了深入的剖析[Univ. of Minnesota, Dept. of Statistics, Reports 113 及 114 (November, 1968); *Revue Fransçaise d' Automatique, Informatique et Recherche Opérationnelle* **6**, R-3(1972 年 12 月), 23～68]。

Sobel 构造了图 42 的过程,它平均仅仅使用 $6\frac{1}{2}$次比较就求得了 6 个元素的第二个最大者。在最坏的情况下,需要进行 8 次比较,而这比 $V_2(6)=7$ 要坏;事实上,D. Hoey 所做的一个穷尽的计算机查找已经证明,对于这个问题的最好的过程,如果限定至多做 7 次比较,则平均使用 $6\frac{26}{45}$次比较。于是,在同时地极小化极大值和极小化平均值两种意义下,似乎没有求 6 个元素的第二个最大元素的最优过程。

令 $\overline{V}_t(n)$表示为求 n 个元素的第 t 个最大者所需要的极小平均比较数。如同 D. Hoey 所计算的那样,表 2 示出了对于小的 n 的一些精确值。

表 2 进行选择所需的极小平均比较次数

n	$\overline{V}_1(n)$	$\overline{V}_2(n)$	$\overline{V}_3(n)$	$\overline{V}_4(n)$	$\overline{V}_5(n)$	$\overline{V}_6(n)$	$\overline{V}_7(n)$
1	0						
2	1	1					
3	2	$2\frac{2}{3}$	2				
4	3	4	4	3			
5	4	$5\frac{4}{15}$	$5\frac{13}{15}$	$5\frac{4}{15}$	4		
6	5	$6\frac{1}{2}$	$7\frac{7}{18}$	$7\frac{7}{18}$	$6\frac{1}{2}$	5	
7	6	$7\frac{149}{210}$	$8\frac{509}{630}$	$9\frac{32}{105}$	$8\frac{509}{630}$	$7\frac{149}{210}$	6

R. W. Floyd 在 1970 年发现,n 个元素的中值,平均说来,可以仅仅通过$\frac{3}{2}n+O(n^{2/3}\log n)$次比较求得。几年后,他和 R. L. Rivest 改进了这个方法并且构造了一个优美的算法来证明

$$\overline{V}_t(n)\leqslant n+\min(t,n-t)+O(\sqrt{n\log n}) \tag{16}$$

(参见习题 13 和 29)。

根据对 $t=2$ 的一种 Sobel 构造的推广,David W. Matula[*Washington Univ. Tech. Report* AMCS-73-9(1973)]用另一种方法证明

$$\overline{V}_t(n)\leqslant n+t\lceil\lg t\rceil(11+\ln\ln n) \tag{17}$$

因此，对于固定的 t，平均的工作量可减少到仅仅用 $n + O(\log\log n)$ 次比较。对于 $\overline{V}_t(n)$ 的一个优美的下限见于习题 25。

排序和选择问题是一个更具一般性问题的特殊情况，该问题即：求同一个给定的偏序相一致的 n 个给定元素的一个排列。A. C. Yao[*SICOMP* **18**(1989)，679～689]已经证明，如果偏序是由有 n 个顶点和 k 个连通分量的一个无回路有向图 G 定义的，则为解决这样一个问题，在最坏情况下和平均情况下，所需要的极小比较次数，总是 $\Theta(\lg(n!\,/T(G) + n - k)$，其中 $T(G)$ 是同偏序一致的排列的总个数（G 的拓朴排序的个数）。

习　题

1.[*15*] 在 Lewis Carroll 的锦标赛中（图 39 和图 40），为什么选手 13 被淘汰，尽管在第三轮中他赢了？

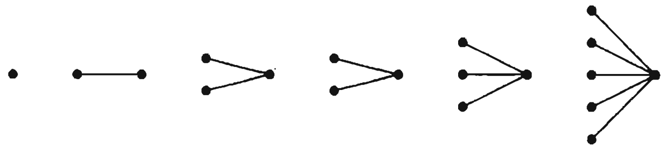
▶2.[*M25*] 证明，在通过一系列的比较，找出了 n 个元素的第 t 个最大者之后，我们还知道哪 $t-1$ 个元素是大于它的，哪 $n-t$ 个元素是小于它的。

3.[*20*] 证明，对于 $1 \leqslant t < n$，$V_t(n) > V_t(n-1)$ 和 $W_t(n) > W_t(n-1)$。

4.[*M25*]（F. Fussenegger 和 H. N. Gabow）证明 $W_t(n) \geqslant n - t + \lceil \lg n^{\underline{t-1}} \rceil$。

5.[*10*] 证明 $W_3(n) \leqslant V_3(n) + 1$。

6.[*M26*]（R. W. Floyd）给定 n 个不同的元素 $\{X_1, \cdots, X_n\}$ 以及某些 (i, j) 对的关系 $X_i < X_j$ 的一个集合，我们希望找出第二个最大的元素。如果已经知道对 $j \neq k$ 有 $X_i < X_j$ 及 $X_i < X_k$，则 X_i 不可能是第二个最大的元素，所以它可被消去。得到的关系有一个如同



的形式，即，可以由一个多重集合 $(l_1, l_2, \cdots, l_m)$ 表示的 m 组元素；第 j 组包含 $l_j + 1$ 个元素，已知它们当中的一个大于其它的。例如，上边的配置可以由多重集合(0,1,2,2,3,5)来描述；当不知道任何关系时，我们有 n 个零的一个多重集合。

令 $f(l_1, l_2, \cdots, l_m)$ 是为找出这样一个偏序集合的第二个最大的元素所需要的极小比较数，证明

$$f(l_1, l_2, \cdots, l_m) = m - 2 + \lceil \lg (2^{l_1} + 2^{l_2} + \cdots + 2^{l_m}) \rceil$$

[提示：证明，最好的策略总是比较两个最小组的最大元素，直到把 m 归结成 1 为止；对 $l_1 + l_2 + \cdots + l_m + 2m$ 使用归纳法]。

7.[*M20*] 证明(8)。

8.[*M21*] Kislitsyn 公式(6)是以使用具有 n 个外部节点的完备二叉树进行树选择排序为基础的。对任何 t 和 n，以某个其它树为基础的一个树选择算法将给出更好的上限吗？

▶9.[*20*] 使用 Hadian 和 Sobel 的替代选择方法[参见(11)]，画出在至多 6 步中即找出 5 个元素的中值的比较树。

10.[*35*] 证明可以在至多 10 步之内求出 7 个元素的中值。

11.[*38*](K. Noshita) 证明可以在至多 14 步之内求出 9 个元素的中值，而这 14 步的前 7 步和 Doren 的方法一样。

12.[*21*](Hadian 和 Sobel) 证明：$V_3(n) \leqslant V_3(n-1)+2$[提示：由去掉$\{X_1, X_2, X_3, X_4\}$之最小者开始]。

13.[*HM28*](R. W. Floyd) 证明，如果使用一个递归定义的方法，从找出$\{X_1, \cdots, X_{n^{2/3}}\}$的中值元素开始，则可以通过平均$\frac{3}{2}n+O(n^{2/3}\log n)$次比较找出$\{X_1, \cdots, X_n\}$的中值。

▶14.[*20*](M. Sobel) 设 $U_t(n)$是为找出 n 个元素的第 t 个最大者(但无须知道它们的相对次序)，所需要的极小比较数。证明 $U_2(5) \leqslant 5$。

15.[*22*](I. Pohl) 假设我们对空间(而不是时间)的极小化感兴趣。如果每个元素都填入一个字，而且如果这些元素逐个地输入到一个寄存器中，则为了计算 n 个元素的第 t 个最大者，至少需要多少存储单元？

16.[*25*](I. Pohl) 证明，至多使用$\lceil \frac{2}{3}n \rceil - 2$ 次比较，便可以找出 n 个元素的集合的极大和极小；而且这个比较数不可能被降低[提示：这样一个算法的任何阶段都可表示作一个四元组(a, b, c, d)，其中 a 元素尚未被比较，b 已经获胜过但是尚未失败过，c 已经失败过而尚未获胜过，d 有胜有负。试构造一个适当的敌手]。

17.[*20*](R. W. Floyd) 试证使用至多$\lceil \frac{3}{2}n \rceil - k - l + \Sigma_{n+1-k<j\leqslant n}\lceil \lg j \rceil + \Sigma_{n+1-l<j\leqslant n}\lceil \lg j \rceil$次比较，就有可能依次地选择 n 个元素的一个集合的 k 个最大和 l 个最小的元素。

18.[*M20*] 如果在定理 L 的证明中使用了大小为 5，而不是 7 的组，则将得到什么定理？

19.[*M42*] 把表 2 扩充到 $n=8$。

20.[*M47*] 当 $n \to \infty$ 时，$\overline{V}_2(n)-n$ 的渐近值是多少？

21.[*32*] (P. V. Ramanan 和 L. Hyafil) 证明，当$k \geqslant t \geqslant 2$ 时，$W_t(2^k+2^{k+1-t}) \leqslant 2^k+2^{k+1-t}+(t-1)(k-1)$；并且证明，由于习题 4，等式对于无穷多个 k 和 t 成立[提示：保持两个淘汰树并且适当地合并它们的结果]。

22.[*24*](David G. Kirkpatrick) 试证明，当 $4\cdot 2^k < n-1 \leqslant 5\cdot 2^k$ 时，对于 $V_3(n)$的上限(11)可减 1 如下：(i)构造大小为 2^k 的 4 个淘汰树。(ii)求 4 个极大值的极小，并抛弃它的树的所有 2^k个元素。(iii)使用已知信息，构造大小为 $n-1-2^k$ 的一棵淘汰树。(iv)像在(ii)的证明中那样继续。

23.[*M49*] 当 $n \to \infty$时，$V_{\lceil n/2 \rceil}$的渐近值是多少？

24.[*HM40*] 试证明，对于 $t \leqslant \lceil n/2 \rceil$，$\overline{V}_t(n) \leqslant n+t+O(\sqrt{n \log n})$。提示：证明，通过这许多的比较，我们事实上可以求出第$(t-\sqrt{t \ln n})$和第$\lceil t+\sqrt{t \ln n} \rceil$个元素，在这之后就可容易地确定第 t 个的位置。

▶25.[*M35*] (W. Cunto 和 J. I. Munro) 试证明，当 $t \leqslant \lceil n/2 \rceil$ 时，$\overline{V}_t(n) \geqslant n+t-2$。

26.[*M32*](A. Schönhage, 1974) (a)使用习题 14 的符号，证明对于 $n \geqslant 3$，$U_t(n) \geqslant \min(2+U_t(n-1), 2+U_{t-1}(n-1))$[提示：只要当前的偏序不是完全由具有·或 ·—· 形式的分量组成的，就由 n 减小成 $n-1$，由此构造出一个敌手]。

(b)类似地，对于 $n \geqslant 5$，通过构造处理分量·，·—·，， 的一个敌手，证明

$$U_t(n) \geqslant \min(2+U_t(n-1), 3+U_{t-1}(n-1), 3+U_t(n-2))$$

(c)因此，对于 $1\leqslant t\leqslant n/2$，$U_t(n)\geqslant n+t+\min(\lfloor(n-t)/2\rfloor,t)-3$[当 V 或 W 代替 U 时，在(a)和(b)中的不等式也适用，因此建立了表 1 中的若干个表项的最优性]。

27.[*M34*] 随机化敌手是一个这样的敌手算法，当它进行决策时，允许它使用硬币投掷。

(a)设 A 是一个随机化敌手，并令 $\Pr(l)$是 A 达到一个给定的比较树的叶 l 的概率。试证明，如果对于所有的 l，$\Pr(l)\leqslant p$，则这个比较树的高度$\geqslant\lg(1/p)$。

(b)给定有待在稍后选择的整参数 q 和 r，对于选择 n 个元素第 t 个最大者的问题，考虑下列敌手：

A1. 选择 t 个元素的一个随机集合 T；所有$\binom{n}{t}$种可能性是同等可能的(我们将确保 $t-1$ 个最大元素属于 T)。设 $S=\{1,\cdots,n\}\setminus T$ 是其它元素，并置 $S_0\leftarrow S$，$T_0\leftarrow T$；S_0 和 T_0 将表示那些可能变成第 t 个最大的元素。

A2. 当 $|T_0|>r$ 时，判定所有比较 $x:y$ 如下：如果 $x\in S$ 和 $y\in T$，就说，$x<y$。如果 $x\in S$ 和 $y\in S$，掷一次硬币来判定，如果较小的元素来自 S_0 就从 S_0 中删去这个元素。如果 $x\in T$ 和 $y\in T$，掷一次硬币来判定，如果较大者来自 T_0 就从 T_0 删去这个元素。

A3. 只要 $|T_0|=r$，就把这些元素分为 P,Q,R 3 类，如下：如果 $|S_0|<q$，令 $P=S$，$Q=T_0$，$R=T\setminus T_0$。否则，对于每个 $y\in T_0$，令 $C(y)$是已经同 y 做过比较的 S 的元素，并选择 y_0 使得 $|C(y_0)|$ 是极小。令 $P=(S\setminus S_0)\cup C(y_0)$，$Q=(S_0\setminus C(y_0))\cup\{y_0\}$，$R=T\setminus\{y_0\}$。通过指出 P 的元素小于 Q 的元素，和 Q 的元素小于 R 的元素，来判定所有未来的比较；当 x 和 y 为同一类时，掷一次硬币。

证明，如果 $1\leqslant r\leqslant t$ 和在步骤 A3 开始时 $|C(y_0)|\leqslant q-r$，则每个叶被达到的概率$\leqslant(n+1-t)/\left(2^{n-q}\binom{n}{t}\right)$。提示：证明至少投掷了 $n-q$ 次硬币。

(c)继续(b)，试证对于所有整数 q 和 r，我们有

$$V_t(n)\geqslant\min\left(n-1+(r-1)(q+1-r),n-q+\lg\left(\binom{n}{t}/(n+1-t)\right)\right)$$

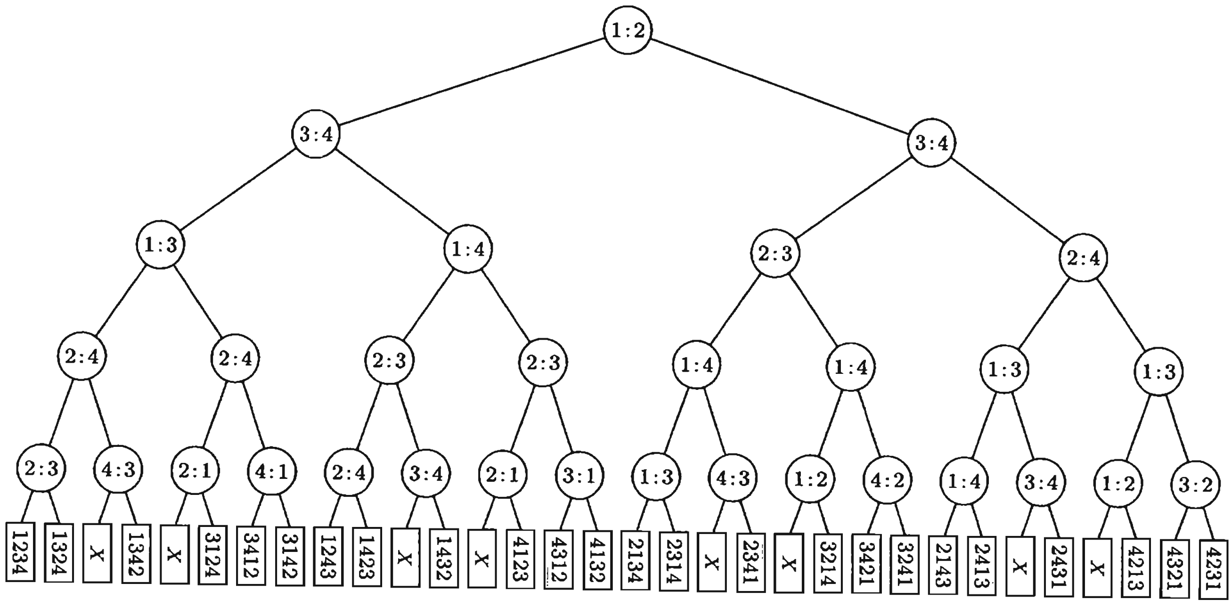
(d)通过选择 q 和 r 来建立(14)。

*5.3.4 排序网络

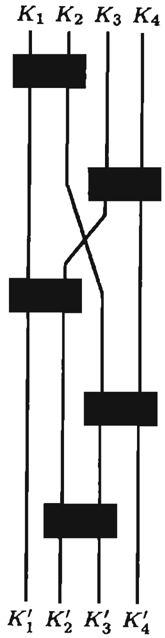
在本小节中，我们将研究一种受限制类型的排序，由于它的应用和丰富的理论基础使它特别有趣。新的限制是坚持齐性的比较序列。齐性的定义是：每次比较 K_i 和 K_j 后，随后的比较对于 $K_i<K_j$ 和 $K_i>K_j$ 是完全一样的，仅将 i 和 j 互换即可。

图 43(a)示出了一株满足这个齐性条件的比较树。注意，每层都有同样多次的比较。所以在进行了 m 次比较后，有 2^m 个结果。由于 n! 不是 2 的幂，所以某些比较必然是多余的。换句话说，这个树的某些分支必须做多于实际需要的比较，才能确保这株树所有对应的分支都恰当地排好序。这里“多余”的意思是，它们的子树之一实际上是不可能出现的。

由于这样一株树被从顶部到下部的每条路径惟一地确定，所以这样一个排序方案最容易表示成一个网络；参见图43(b)。在这样一个网络中，方框表示“比较模



(a)



(b)

图 43

(a)齐性的比较树;(b)对应的网络。

块”,它有两个输入(表示成从上边进入到这个模块的直线)和两个输出(表示引向下边的直线);左边的输出是两个输入中的较小者,而右边的输出是较大者。在这个网络的下部,K'_1 是 $\{K_1, K_2, K_3, K_4\}$ 的最小者,K'_2 是第二个最小者,等等。不难证明,任何排序网络都对应于在上述意义下的齐性的比较,而任何齐性树都对应于比较模块的一个网络。

附带说一句,我们要指出,从工程的观点来看,比较模块是相当容易制造的。例如,假设直线包含二进制数,其中每一个单位时间有一个二进位数进入每个模块,最先进入的是最高位。每个比较模块都有 3 个状态,并按如下方式进行:

时间 t		时间 $(t+1)$	
状态	输入	状态	输出
0	0 0	0	0 0
0	0 1	1	0 1
0	1 0	2	0 1
0	1 1	0	1 1
1	x y	1	x y
2	x y	2	y x

开始时,所有的模块都在状态 0 并且输出 0 0。一个模块只要它的输入不同就进入状态 1 或状态 2。如果在直线 K'_1 和 K'_4 上附加一个适当的延迟元件,则在时间 t 时在图 43(b)顶上开始送来的数,将被排好次序,并从 $t+3$ 时开始由底下输出。

为了建立排序网络的理论,不妨以一种如图 44 中所示的稍微不同的方式来表示它们。这里,数从左边进入,并用两条直线间的垂直连接表示比较模块;每个比较块必要时交换其输入量,使得在通过这个比较块之后较大的数出现在下边的线上。在这个图形的右边,所有的数都是从上到下有序的。

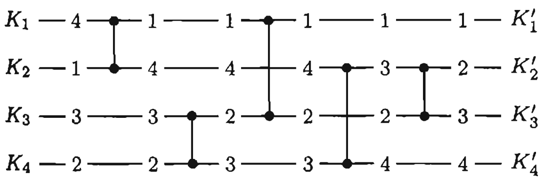


图 44 当对 4 个数的序列⟨4,1,3,2⟩排序时表示图 43 的网络的另一种方式

我们以前关于最优排序的研究,集中在使比较次数极小化上,而很少或者根本不注意它引起的任何数据移动或可能需要的判定结构的复杂性。在这方面,排序网络有明显的优点,因为数据可以保留在 n 个地点,而且判定结构是“直线”式的;不需要记住以前比较的结果——因为方案是预先就固定好的。排序网络的另一个重要优点是,有可能重叠若干个操作,同时地执行它们(在一台适当的机器上)。例如,当允许同时的非重叠的比较时,图 43 和图 44 中的 5 步可以叠合成 3 步,因为前 2 步和第二个 2 步可以组合在一起;在本小节稍后将剖析排序网络的这个性质。因此,排序网络可能是非常有用的,尽管还不完全清楚对很大的 n,能否构造有效的 n 个

元素的排序网络；我们将会发现，为了保持齐性的判定结构，需要许多附加的比较。

当给定一个 n 元素的网络时，存在两种简单的方法来构造一个 $n+1$ 个元素的排序网络，或用插入原理，或用选择原理。图 45(a)说明在头 n 个元素已经排序之后，如何把第 $n+1$ 个元素插入到它应有位置中去；而这个图的(b)部分说明在对剩下的一些元素排序之前，如何来选择最大的元素。重复应用图 45(a)，就得到了类似于直接插入排序的网络(算法 5.2.1S)，而重复应用图 45(b)，就产生了类似于冒泡排序的网络(算法 5.2.2B)。图 46 给出了相应的 6 元素的网络。注意，当允许同时操作时，两个方法都归结为同一个“三角形”的有 $2n-3$ 个阶段的过程(图 47)。

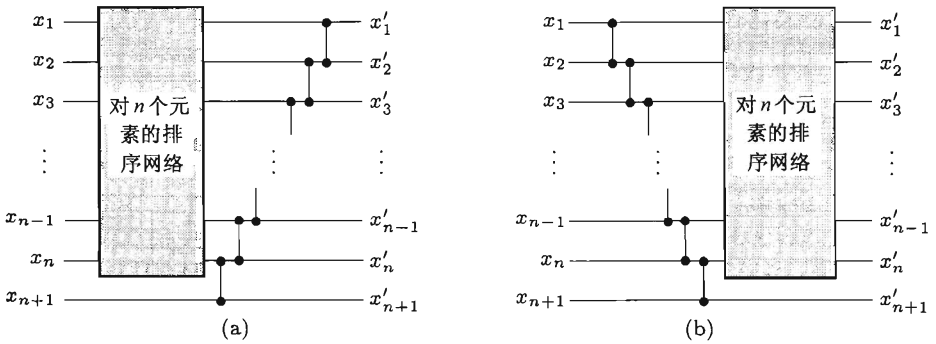


图 45　从 n 排序器构造$(n+1)$排序器

(a)插入；(b)选择。

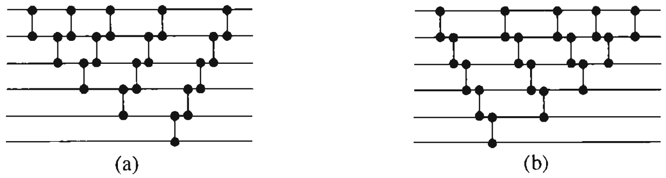


图 46　通过重复应用图 45 得到的类似于初等内部排序图式的网络

(a)直接插入；(b)冒泡排序。

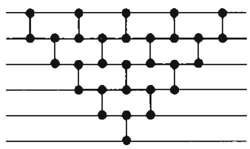
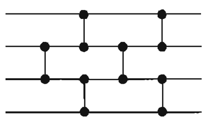
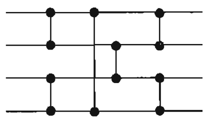


图 47　在并行的情况下直接插入＝冒泡排序！

容易证明：图 43 和图 44 的网络将把 4 个数的任何集合排成为有序的，因为前 4 个比较器把最小和最大的元素放置到正确的位置，最后的比较器把剩下的 2 个元素顺序排序。但是要知道一个给定的网络是否对所有可能的输入序列进行排序，并不总是那样容易的；例如：



和



都是正确的 4 个元素的排序网络，但是它们正确性的证明却非显然。试验 n 个不同元素的所有 $n!$ 个排列的每个 n 元素网络，这肯定是充分的。但是事实上，我们可

以通过少得多的试验获得之:

定理 Z(*0-1 原理*) *如果具有 n 个输入线的一个网络,把 0 和 1 的所有 2^n 个序列排成为非减次序,则它将把 n 个数的任意序列排成为非减的次序。*

证明(这是 Bouricius 定理的特殊情况,习题 5.3.1-12) 如果 $f(x)$是任何单调函数,且每当 $x\leqslant y$ 时,$f(x)\leqslant f(y)$,而且如果一个给定的网络把$\langle x_1,\cdots,x_n\rangle$变换成$\langle y_1,\cdots,y_n\rangle$,则容易看出这个网络将把$\langle f(x_1),\cdots,f(x_n)\rangle$变换成$\langle f(y_1),\cdots,f(y_n)\rangle$。如果对某个 i,$y_i>y_{i+1}$,则考察单调函数 f,它把所有$<y_i$ 的数变成 0,以及把所有$\geqslant y_i$ 的数变成 1;这定义了诸 0 和 1 的一个序列$\langle f(x_1),\cdots,f(x_n)\rangle$,它不被这个网络所排序。因此,如果所有的 0-1 序列被排序,则对于 $1\leqslant i<n$,我们有 $y_i\leqslant y_{i+1}$。 ▌

在构造排序网络中,0-1 原理是十分有帮助的。作为一个非平凡的例子,我们可以推导 Batcher 的"合并交换"(算法 5.2.2M)的一种推广形式。其思想是通过独立地对前 m 个元素和最后 n 个元素排序来实现对全部 $m+n$ 个元素的排序,然后对结果应用(m,n)合并网络。一个(m,n)合并网络可以归纳地构造如下:

a)如果 $m=0$ 或 $n=0$,则这个网络是空的。如果 $m=n=1$,则这个网络是单个比较模块。

b)如果 $mn>1$,令有待合并的诸序列为$\langle x_1,\cdots,x_m\rangle$和$\langle y_1,\cdots y_n\rangle$。合并"奇序列"$\langle x_1,x_3,\cdots,x_{2\lceil m/2\rceil-1}\rangle$和$\langle y_1,y_3,\cdots y_{2\lceil n/2\rceil-1}\rangle$,得到排好序的结果$\langle v_1,v_2,\cdots,v_{\lceil m/2\rceil+\lceil n/2\rceil}\rangle$;合并"偶序列"$\langle x_2,x_4,\cdots,x_{2\lfloor m/2\rfloor}\rangle$和$\langle y_2,y_4,\cdots,y_{2\lfloor n/2\rfloor}\rangle$,得到排好序的结果$\langle w_1,w_2,\cdots,w_{\lfloor m/2\rfloor+\lfloor n/2\rfloor}\rangle$。最后,应用比较交换操作

$$w_1:v_2,\ w_2:v_3,\ w_3:v_4,\ \cdots,\ w_{\lfloor m/2\rfloor+\lfloor n/2\rfloor}:v^* \tag{1}$$

到序列

$$\langle v_1,w_1,v_2,w_2,v_3,w_3,\cdots,v_{\lfloor m/2\rfloor+\lfloor n/2\rfloor},w_{\lfloor m/2\rfloor+\lfloor n/2\rfloor},v^*,v^{**}\rangle \tag{2}$$

上,这个结果将是排好序的。注意,这里如果 m 和 n 都是偶数,则 $v^*=v_{\lfloor m/2\rfloor+\lfloor n/2\rfloor+1}$不存在,另外,除非 m 和 n 都是奇数,否则 $v^{**}=v_{\lfloor m/2\rfloor+\lfloor n/2\rfloor+2}$也不存在。(1)指出的比较模块的总数为$\lfloor(m+n-1)/2\rfloor$。

Batcher 的(m,n)合并网络称为奇偶合并。按照这些原理构造的一个(4,7)合并如图 48 所示。

为了证明这个相当奇特的合并过程实际上是行得通的,当 $mn>1$ 时,我们使用 0-1 原理,对所有的 0 和 1 的序列来测试它。在初始的 m 排序和 n 排序后,对某个 k 和 l,序列$(x_1,\cdots,x_m)$将由 k 个 0 后边接上 $m-k$ 个 1 组成,而序列$\langle y_1,\cdots,y_n\rangle$将是 l 个 0 后边接上 $n-l$ 个 1。因此序列$\langle v_1,v_2,\cdots\rangle$将恰由$\lceil k/2\rceil+\lceil l/2\rceil$个 0,后边接一些 1 组成;而$\langle w_1,w_2,\cdots,\rangle$将由$\lfloor k/2\rfloor+\lfloor l/2\rfloor$个 0,后边接一些 1 组成。现在这里的要点是

$$(\lceil k/2\rceil+\lceil l/2\rceil)-(\lfloor k/2\rfloor+\lfloor l/2\rfloor)=0,1\text{ 或 }2 \tag{3}$$

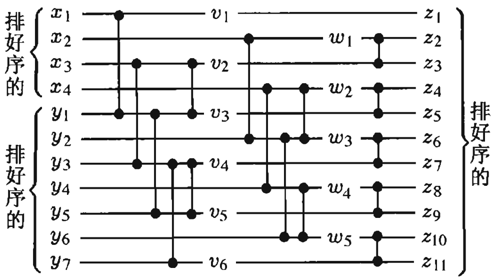


图 48 当 $m=4$ 和 $n=7$ 时的奇偶合并

如果这个差是 0 或 1,则序列(2)已经有序了,如果这个差是 2,则(1)中的比较交换之一就把问题全都解决了。这就完成了证明(注意,0-1 原理把合并问题从 $\binom{m+n}{m}$ 种情况的考虑归结为仅仅$(m+1)(n+1)$种,它们由两个参数 k 和 l 表示)。

设 $C(m,n)$是对于 m 和 n 在奇偶合并中使用的比较模块个数,不计初始的 m 排序和 n 排序,我们有

$$C(m,n)=\begin{cases} mn & \text{如果 } mn\leqslant 1 \\ C(\lceil m/2\rceil,\lceil n/2\rceil)+C(\lfloor m/2\rfloor,\lfloor n/2\rfloor)+\lfloor(m+n-1)/2\rfloor & \text{如果 } mn>1 \end{cases} \tag{4}$$

一般说来,这不是 m 和 n 的特别简单的函数,但是注意 $C(1,n)=n$ 和

$$C(m+1,n+1)-C(m,n)=1+C(\lfloor m/2\rfloor+1,\lfloor n/2\rfloor+1)-C(\lfloor m/2\rfloor,\lfloor n/2\rfloor) \quad \text{如果 } mn\geqslant 1$$

可以导出关系

$$C(m+1,n+1)-C(m,n)=\lfloor \lg m\rfloor+2+\lfloor n/2^{\lfloor \lg m\rfloor+1}\rfloor \quad \text{如果 } n\geqslant m\geqslant 1 \tag{5}$$

因此

$$C(m,m+r)=B(m)+m+R_m(r) \quad \text{对于 } m\geqslant 0,\quad r\geqslant 0 \tag{6}$$

其中 $B(m)$是等式 5.3.1-(3)的"二叉插入"函数 $\sum_{k=1}^{m}\lceil \lg k\rceil$,而其中 $R_m(r)$表示下列级数的前 m 项之和

$$\left\lfloor\frac{r+0}{1}\right\rfloor+\left\lfloor\frac{r+1}{2}\right\rfloor+\left\lfloor\frac{r+2}{4}\right\rfloor+\left\lfloor\frac{r+3}{4}\right\rfloor+\left\lfloor\frac{r+4}{8}\right\rfloor+\cdots+\left\lfloor\frac{r+j}{2^{\lfloor \lg j\rfloor+1}}\right\rfloor+\cdots \tag{7}$$

特别是,当 $r=0$ 时,我们有重要的特殊情况

$$C(m,m)=B(m)+m \tag{8}$$

进而,如果 $t=\lceil \lg m\rceil$,则

$$\begin{aligned} R_m(r+2^t)&=R_m(r)+1\cdot 2^{t-1}+2\cdot 2^{t-2}+\cdots+2^{t-1}\cdot 2^0+m \\ &=R_m(r)+m+t\cdot 2^{t-1} \end{aligned}$$

因此 $C(m,n+2^t)-C(m,n)$有一个简单的形式,并且

$$C(m,n)=\left(\frac{t}{2}+\frac{m}{2^t}\right)n+O(1)\quad 对固定的\ m,n\to\infty,t=\lceil \lg m\rceil \tag{9}$$

$O(1)$项是 n 的一个最终周期函数,其周期长度为 2^t。由等式(8)和习题 5.3.1-15,当 $u\to\infty$时,$C(n,n)=n\lg n+O(n)$。

极小比较网络 令 $\hat{S}(n)$为 n 个元素的排序网络中所需要的比较器的极小个数;显然 $\hat{S}(n)\geqslant S(n)$,其中 $S(n)$为一个不需齐性的排序过程中所需要的极小比较次数(参见 5.3.1 小节)。我们有 $\hat{S}(4)=5=S(4)$,所以当 $n=4$ 时,新的限制并不引起效率的损失;但当 $n=5$ 时,结果已经是 $\hat{S}(5)=9$ 而 $S(5)=7$。确定 $\hat{S}(n)$的问题似乎比确定$S(n)$的问题更为困难;甚至连 $\hat{S}(n)$的渐近特性也还不知道。

回顾这个问题的历史是有趣的,因为这里迈出的每一步都是很艰难的。大约在 1954 年,P. N. Armstrong, R. J. Nelson 以及 D. J. O'Connor 首先剖析了排序网络[见 *U. S. Patent* 3029413];用其专利律师的话说,"通过使用这个技巧,有可能使用较小的双线排序开关来设计经济的 n 线排序开关"。在发现 $\hat{S}(n+1)\leqslant\hat{S}(n)+n$ 之后,他们给出了对于 $4\leqslant n\leqslant 8$ 的特殊构造,分别使用了 5,9,12,18 和 19 个比较器。随后,Nelson 同 R. C. Bose 合作,证明了对于所有的 n,$\hat{S}(2^n)\leqslant 3^n-2^n$;因此 $\hat{S}(n)=O(n^{\lg 3})=O(n^{1.585})$。Bose 和 Nelson 在 *JACM* **9**(1962),282～296 上发表了他们有趣的方法,文中他们推测它是最好的。T. N. Hibbard[*JACM* **10**(1963),142～150] 发现了一个类似但稍简单些的构造,它使用相同的比较数,由此更增强了这个推测。

1964 年,R. W. Floyd 和 D. E. Knuth 发现了解决这个问题的新方法,导致了形如$\hat{S}(n)=O(n^{1+c/\sqrt{\log n}})$的一个渐近限值。K. E. Batcher 独立地发现了上边概述的一般的合并技巧;使用了由递推式

$$c(1)=0,c(n)=c(\lceil n/2\rceil)+c(\lfloor n/2\rfloor)+C(\lceil n/2\rceil,\lfloor n/2\rfloor)\quad n\geqslant 2 \tag{10}$$

对于定义的比较器个数,他证明(见习题 5.2.2-14)

$$c(2^t)=(t^2-t+4)2^{t-2}-1$$

而且由此得出 $\hat{S}(n)=O(n(\log n)^2)$。Batcher, Floyd 及 Knuth 都过了一段时间才发表他们的构造[*Notices of the Amer. Math. Soc.* **14**(1967),283;*Proc. AFIPS Spring Joint Computer Conf.* **32**(1968),307～314]。

一些人已经找出方法来改进 Batcher 的合并交换的构造所用的比较次数;下表说明当前已知的 $\hat{S}(n)$的最好上限:

$n=$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$c(n)=$	0	1	3	5	9	12	16	19	26	31	37	41	48	53	59	63
$\hat{S}(n)\leqslant$	0	1	3	5	9	12	16	19	25	29	35	39	45	51	56	60

(11)

由于对 $8<n\leqslant 16$,$\hat{S}(n)<c(n)$,故对所有 $n>8$,合并交换是非最优的。当 $n\leqslant 8$ 时,合并交换使用的比较器个数与Bose和Nelson的构造相同。Floyd和Knuth于

1964～1966 年证明了当 $n\leqslant 8$ 时，$\hat{S}(n)$的上列值是精确的[见*A Survey of Combinatorial Theory North-Holland* ,1973),163～172]；当 $n>8$ 时，$\hat{S}(n)$的值仍然是未知的。

图 49 所示为导致(11)中的值的构造。以有趣的三路合并为基础的 $n=9$ 的网络，是 1964 年由 R.W.Floyd 建立的；它的正确性可以通过使用习题 27 中描述的一般原理来证明。1969 年，A.Waksman 把输入当作{1,2,…,10}的排列，并试图尽可能地减少在每个阶段中，每一直线上出现的可能的数值的数目，同时保留某种对称性，由此发现了 $n=10$ 的网络。

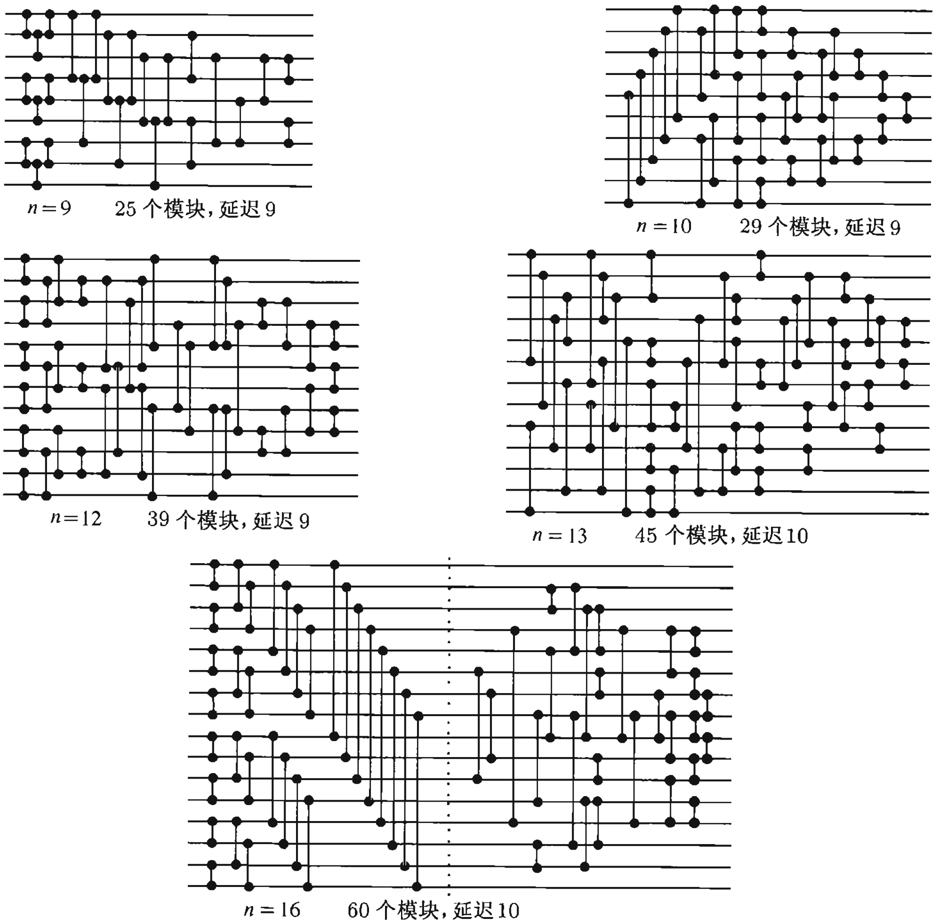


图 49 有效的排序网络

对于 $n=13$ 所示的网络有十分不同的家谱：Hughes Juillé[*Lecture Notes in Comp. Sci.* **929** (1995),246～260]使用一个计算机程序，通过模拟基因的进化过程

来构造它。这个网络没有明显的意义，但它奏效——而且它比迄今根据人们的推理所设计的任何其他构造都更短些。

1969 年，G. Shapiro 发现了对于 16 个元素的 62 个比较器的排序网络，这是相当令人惊讶的。因为当 n 是 2 的乘方时，Batcher 的方法（63 个比较）看起来是最好的。在听说 Shapiro 的构造后不久，M. V. Green 由于找出图 49 中包含 60 个比较的排序器，而使人们更为惊讶。Green 构造的前一部分是容易理解的；在对虚线左边进行了 32 次比较/交换之后，这些直线可以以 $\{a,b,c,d\}$ 的 16 个子集按下面的方式来标号：每当 s 是 t 的一个子集时，标号为 s 的直线包含一个数，这个数小于或等于标号为 t 的直线的内容。习题 32 中进一步讨论了这里的排序状态。然而，在 Green 的网络的随后层次上所做的比较变得越来越神秘，而且至今未有人看出怎样来推广这个构造，以便相应地得到对于 n 的更高值的有效网络。

Shapiro 和 Green 也发现了 $n=12$ 的网络。当 $n=11,13,14$ 或 15 时，通过把 $n+1$ 网络中最底下的线和触及该线的所有比较器一起撤销，就可以建立好的网络。

由 D. Van Voorhis 给出的对于 256 个元素的当前已知为最好的排序网络，Batcher 方法的 3839 相比，其证明 $\hat{S}(256)\leqslant 3651$［参见 R. L. Drysdale 和 F. H. Young, *SICOMP* **4** (1975), 264～270］。当 $n\to\infty$ 时，事实上结果为 $\hat{S}=O(n\log n)$；这个令人吃惊的上限是由 Ajtai, Komlos 和 Szemerédi 在 *Combinatorica* **3**(1983), 1～19 中证明的。他们所构造的网络没有实际意义，因为引进许多的比较器只不过是为了节省 $\log n$ 的一个因子，除非 n 超过地球上所有计算机的总的内存容量，否则 Batcher 的方法要好得多。但是 Ajtai, Komlos 和 Szemerédi 的定理确实建立了一直到一个常数因子的 $\hat{S}(n)$ 的真正渐近增长比率。

极小时间网络　在排序网络的物理实现中，以及在并行计算机上，有可能同时进行非重叠的比较交换；因此自然地提出了延迟时间极小化的问题。稍做考虑就可看出，如果把一条路径定义为任何自左到右的路线，这个路线可能在比较器处改换所走的直线，则一个排序网络的延迟时间，等于通过网络与任何“路径”接触的比较器之极大个数。可以在每个比较器上放置一个序列编号，来指出它可以被执行的最早时刻；这个编号比在它的输入线上较早出现的比较器序列编号的极大值大 1（见图 50(a)；此图的(b)部分示出重画的同一网络，其中每一个比较完成于最早可能的时刻）。

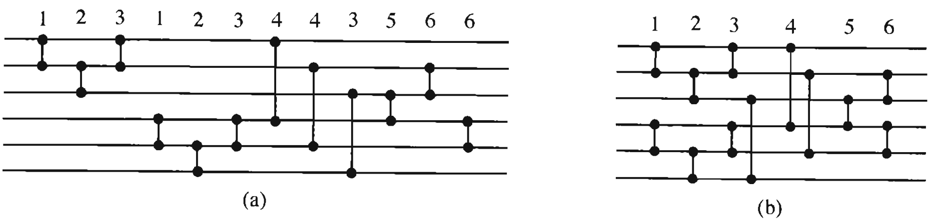


图 50　在最早可能的时刻进行每个比较

上面所描述的 Batcher 奇偶合并网络花费 $T_B(m,m)$个时间单位，其中 $T_B(m,0)=T_B(0,n)=0$，$T_B(1,1)=1$，且

$$T_B(m,n)=1+\max(T_B(\lfloor m/2\rfloor,\lfloor n/2\rfloor),T_B(\lceil m/2\rceil,\lceil n/2\rceil)\quad 对于\ mn\geqslant 2$$

我们可以使用这些关系，通过归纳法来证明 $T_B(m,n+1)\geqslant T_B(m,n)$；因此对于 $mn\geqslant 2$，$T_B(m,n)=1+T_B(\lceil m/2\rceil,\lceil n/2\rceil)$，并由此得出

$$T_B(m,n)=1+\lceil\lg\max(m,n)\rceil\qquad 对于\ mn\geqslant 1 \tag{12}$$

由此，习题 5 证明，Batcher 排序方法的延迟时间为

$$\binom{1+\lceil\lg n\rceil}{2} \tag{13}$$

令 $\hat{T}(n)$是在 n 个元素的任何排序网络中，可达到的极小延迟时间。有可能来改进上面描述的某些网络，使得它们有较少的延迟时间，但不使用更多的比较器，如图 51 中对于 $n=6$，$n=9$ 和 $n=11$ 以及习题 7 对于 $n=10$ 所示的那样。如果如图 51 中所示对于 $n=10$、12 和 16 的值得注意的网络那样，我们增加一个或两个额外的模块，则还可以达到更小的延迟时间。这些构造，对于小的 n，产生了 $\hat{T}(n)$的下列上限

$$\begin{array}{lcccccccccccccccc} n= & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16\\ \hat{T}(n)\leqslant & 0 & 1 & 3 & 3 & 5 & 5 & 6 & 6 & 7 & 7 & 8 & 8 & 9 & 9 & 9 & 9\end{array} \tag{14}$$

对于 $n\leqslant 10$，这里给出的值已知是精确的（见习题 4）。图 51 中的网络值得仔细研究，因为它们的排序能力决不是一眼能看出来的；有些是在 1969 年—1971 年间由 G. Shapiro（对于 $n=6,12$）和 D. Van Voorhis（$n=10,16$）发现的。其余的（$n=9,11$）是 Loren Schwiebert 于 2001 年使用遗传方法发现的。

合并网络 令 $\hat{M}(m,n)$表示在一个网络中所需要的极小比较模块数，这个网络把 m 个元素 $x_1\leqslant\cdots\leqslant x_m$ 同 n 个元素 $y_1\leqslant\cdots<y_n$ 加以合并以形成有序序列 $z_1\leqslant\cdots\leqslant z_{m+n}$。现在，还没有发现比上面描述的奇偶合并更优越的合并网络，因此(6)中的函数 $C(m,n)$就表示对于 $\hat{M}(m,n)$已知的最好上限。

R. W. Floyd 已经发现了求这个合并问题下限的一种有趣方法。

定理 F *对于所有* $n\geqslant 1$，$\hat{M}(2n,2n)\geqslant 2\hat{M}(n,n)+n$。

证明 考虑具有 $\hat{M}(2n,2n)$个比较模块的网络，它有能力对所有的输入序列$(z_1,\cdots,z_{4n})$排序，使得 $z_1\leqslant z_3\leqslant\cdots\leqslant z_{4n-1}$和 $z_2\leqslant z_4\leqslant\cdots\leqslant z_{4n}$。可以假定每个模块都对某个 $i<j$ 以$(\min(z_i,z_j),\max(z_i,z_j))$代替$(z_i,z_j)$（见习题 16）。因此诸比较器可以分成 3 类：

a) $i\leqslant 2n$ 和 $j\leqslant 2n$。

b) $i>2n$ 和 $j>2n$。

c) $i\leqslant 2n$ 和 $j>2n$。

类 a)必须至少包含 $\hat{M}(n,n)$个比较器，因为 $z_{2n+1},z_{2n+2},\cdots,z_{4n}$当开始合并时可能已经处于它们最后的位置了；类似地，在类 b)中至少有 $\hat{M}(n,n)$个比较器，进而，

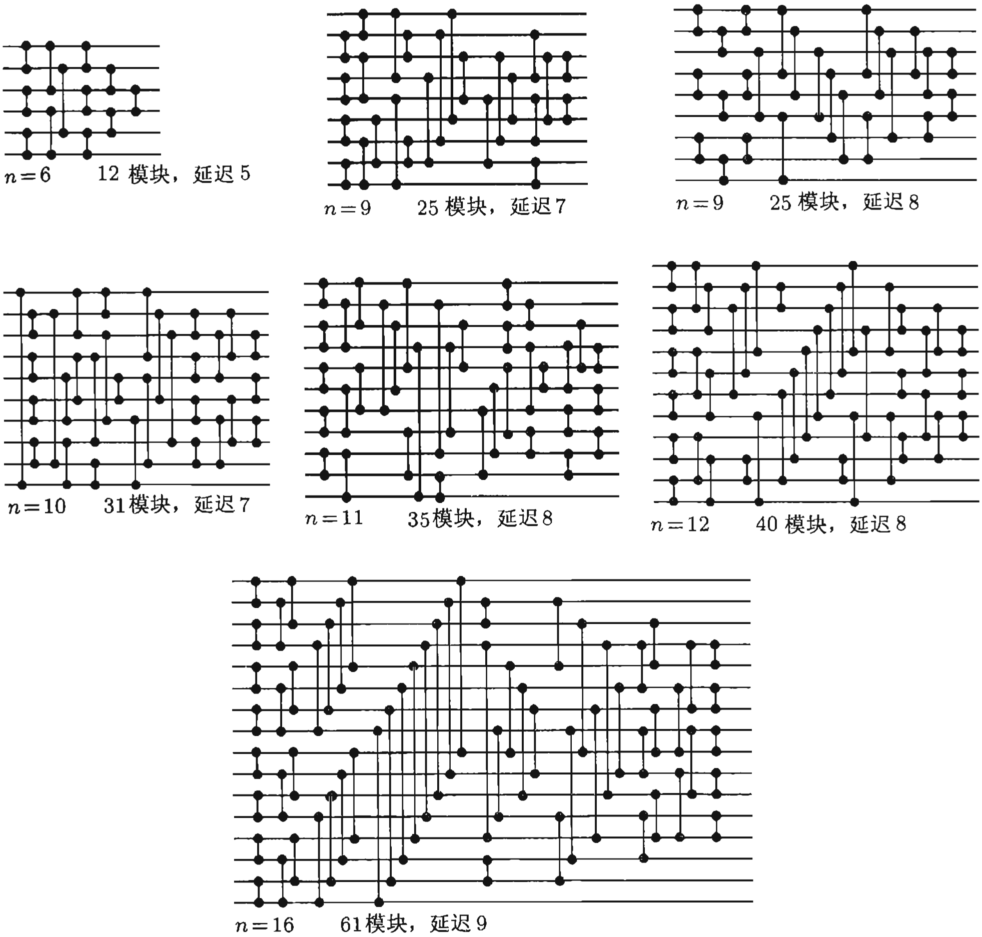


图 51　当并行地执行比较时，非常快的排序网络

输入序列 $\langle 0,1,0,1,\cdots,0,1\rangle$ 说明类 c)至少包含 n 个比较器，因为 n 个 0 必须从 $\{z_{2n+1},\cdots,z_{4n}\}$ 移到 $\{z_1,\cdots,z_{2n}\}$。 ▌

重复使用定理 F 可证明 $\hat{M}(2^n,2^m)\geqslant\frac{1}{2}(m+2)2^m$；因此 $\hat{M}(n,n)\geqslant\frac{1}{2}n\lg n+O(n)$。由定理 5.3.2M 可知，没有网络限制的合并，只需要 $M(n,n)=2n-1$ 个比较；因此我们已经证明，通过网络进行合并本质上比通常的合并更困难。

奇偶合并表明

$$\hat{M}(m,n)\leqslant C(m,n)=\frac{1}{2}(m+n)\lg\min(m,n)+O(m+n)$$

P. B. Miltersen，M. Paterson 和 J. Tarui[*JACM* **43**(1996)，147～165]通过确立下限

$$\hat{M}(m,n) \geqslant \frac{1}{2}((m+n)\lg(m+1) - m/\ln 2) \quad 对于 1 \leqslant m \leqslant n$$

改进了定理 F。因此

$$\hat{M}(m,n) = \frac{1}{2}(m+n)\lg\min(m,n) + O(m+n)$$

A. C. Yao(姚期智)和 F. F. Yao(姚储枫)[*JACM* **23** (1976),566～571]已经证明精确的公式 $\hat{M}(2,n) = C(2,n) = \left\lceil \frac{3}{2}n \right\rceil$。对于 $m = n \leqslant 5$,也已经知道 $\hat{M}(m,n)$ 的值等于 $C(m,n)$;参见习题 9。

双调排序 当允许进行同时的比较时,在等式(12)中已经看到,当 $1 \leqslant m \leqslant n$ 时,奇偶合并使用 $\lceil \lg(2n) \rceil$ 个延迟时间单位。Batcher 已经想出了用于合并的另一类型的网络,称做双调排序器。尽管它要求更多的比较模块,但它把延迟时间降低到 $\lceil \lg(m+n) \rceil$[参见*U. S. Patent* 3428946(1969)]。

我们说 p 个数的一个序列 $\langle z_1, \cdots, z_p \rangle$ 是双调的,如果对于某个 k,$1 \leqslant k \leqslant p$,$z_1 \geqslant \cdots \geqslant z_k \leqslant \cdots \leqslant z_p$(请对照通常的"单调"序列的定义)。一个 p 阶的双调排序器是一个有能力把长度为 p 的任何双调序列,排成为非减次序的比较网络。把 $x_1 \leqslant \cdots < x_m$ 同 $y_1 \leqslant \cdots \leqslant y_n$ 合并的问题,是双调排序问题的一个特殊情况,因为合并可以通过对序列 $\langle x_m, \cdots, x_1, y_1, \cdots, y_n \rangle$ 应用一个 $m+n$ 阶的双调排序器来完成。

注意,若序列 $\langle z_1, \cdots, z_p \rangle$ 是双调的,则它的所有子序列也都是双调的;Batcher 发现了奇偶合并网络之后不久他就发现了:我们可以以任何类似的方式构造一个阶为 p 的双调排序器,方法是首先独立地对双调子序列 $\langle z_1, z_3, z_5, \cdots \rangle$ 和 $\langle z_2, z_4, z_6, \cdots \rangle$ 进行排序,然后比较和变换 $z_1 : z_2$,$z_3 : z_4$(它的证明见习题 10)。如果 $C'(p)$ 是对应的比较模块数,则我们有

$$C'(p) = C'(\lceil p/2 \rceil) + C'(\lfloor p/2 \rfloor) + \lfloor p/2 \rfloor \quad 对于 p \geqslant 2 \tag{15}$$

而且延迟时间显然是 $\lceil \lg p \rceil$。图 52 所示为以这种方式构造的 7 阶双调排序器:它可以用作一个具有 3 个延迟单位的(3,4)或(2,5)合并网络;对于 $m=2$ 和 $n=5$ 的奇偶合并,比较器少一个,但多一级延迟。

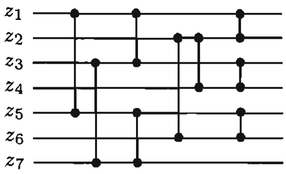


图 52 7 阶 Batcher 双调排序器

2^t 阶的 Batcher 双调排序器是特别有趣的:它由 t 层组成,每层有 2^{t-1} 个比较器。如果我们把输入线编号为 $z_0, z_1, \cdots z_{2^t-1}$,则当且仅当 i 和 j 仅仅在它们的二进位表示的第 l 个最高位上不同时,元素 z_i 才和 l 层上的 z_j 进行比较。这个简单的结构导致了并行排序网络,它像合并交换(即算法 5.2.2M)那样快,但是实现要容易得多(见习题 11 和 13)。

不存在基于同时不相交的比较的并行合并算法,能够在少于 $\lceil \lg(m+n) \rceil$ 阶段中进行排序,无论它是齐性地还是非齐性地工作。在这个意义下,双调合并是最优的(参见习题 46)。在习题 57 中,讨论了以较少的比较,但稍微更复杂的控制逻辑,

来实现这个最优时间的另一个方法。

当 $1\leqslant m\leqslant n$ 时,一个(m,n)合并网络的第 n 个最小的输出依赖于 $2m+[m<n]$个输入(参见习题 29)。如果它能通过有 l 层延迟的比较器来计算,则它至多涉及 2^l 个输入;因此 $2^l\geqslant 2m+[m<n]$,且 $l\geqslant\lceil\lg(2m+[m<n])\rceil$。Batcher 已经证明[Report GER-14122(Akron, Ohio: Goodyear Aerospace Corporation,1968)],如果在这个网络中允许“多重的扇形展开”,即分开诸直线使得在同一时间里同一个数可以输入到许多模块中,则这个极小延迟时间是可以达到的。例如图 53 所示为他给出的 $n=6$ 时,能在仅仅两个延迟级中把 1 个项目同 n 个其它项目合并的网络。当然,具有多个扇形展开的网络不可能符合我们的约定,很容易看出:没有多重扇形展开的任何$(1,n)$合并网络,必须有 $\lg(n+1)$或更多的延迟时间(见习题 45)。

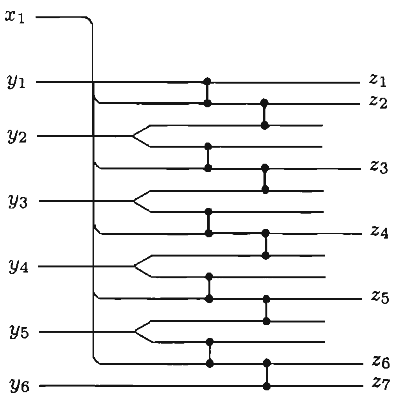


图 53　把 1 个项目同 6 个其它项目合并,用多重扇形展开,以便达到极小延迟时间

选择网络　也可以使用网络来解决 5.3.3 小节的问题。令 $\hat{U}_t(n)$表示在一个网络中所需要的比较器的极小个数,这些比较器把 n 个不同输入中的 t 个最大者送入 t 个指定的输出线;允许它们在这些输出线上以任何次序出现。令 $\hat{V}_t(n)$表示为了把 n 个不同输入中的第 t 个最大者送入一个指定的输出直线,所需要的比较器极小个数;令 $\hat{W}_t(n)$表示把 n 个不同输入中的 t 个最大者以非减次序送到 t 个指定的输出线,所需要的比较器极小个数。不难看出(参见习题 17)

$$\hat{U}_t(n)\leqslant\hat{V}_t(n)\leqslant\hat{W}_t(n) \tag{16}$$

首先假设有 $2t$ 个元素$\langle x_1,\cdots,x_{2t}\rangle$,我们希望选择最大的 t 个;V. E. Alekseev [*Kibernetika* **5**,5(1969),99～103]。已经注意到,要做到这一点我们可以先对$\langle x_1,\cdots,x_t\rangle$和$\langle x_{t+1}\cdots,x_{2t}\rangle$排序,然后比较和交换

$$x_1:x_{2t},\quad x_2:x_{2t-1}\quad\cdots,\quad x_t:x_{t+1} \tag{17}$$

由于这些对中没有一对能包含一个以上的最大的 t 个元素之一(为什么)? Alekseev 过程必定能选择最大的 t 个元素。

如果要选择 nt 个元素中的 t 个最大者,则可应用这个过程 $n-1$ 次,每次消去 t 个元素;因此

$$\hat{U}_t(nt)\leqslant(n-1)(2\hat{S}(t)+t) \tag{18}$$

Alekseev 也对这个选择问题推导出一个有趣的下限。

定理 A　$\hat{U}_t(n)\geqslant(n-t)\lceil\lg(t+1)\rceil$。

证明　考虑选择最小的 t 个元素这样一个等价的问题是最方便的。可以把数

(l,u)附加到一个比较器网络的每条直线上，如图 54 所示，其中 l 和 u 分别表示当

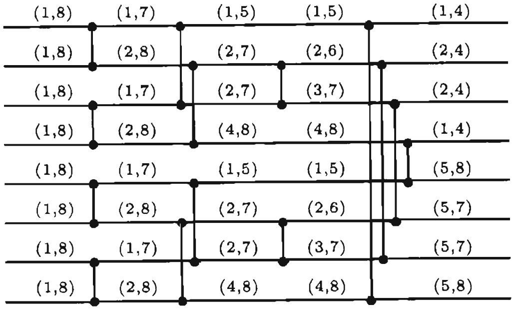


图 54　分开最小的 4 个与最大的 4 个(在这些直线上的数用于定理 A 的证明)

输入是$\{1,2,\cdots,n\}$的一个排列时在该位置可以出现的极小值和极大值。令 l_i 和 l_j 是进行比较 $x_i:x_j$ 之前在直线 i 和 j 上的下限，并令 l_i' 和 l_j' 是进行比较之后对应的下限。显然，$l_i'=\min(l_i,l_j)$，习题 24 证明了这个并非显而易见的关系

$$l_j' \leqslant l_i + l_j \tag{19}$$

现在让我们重新以另一种方式解释网络的操作(见图 55)：假定所有的输入线都含有零，而且每个“比较器”把它的输入中的较小者置于上边的直线，并把较大者加 1

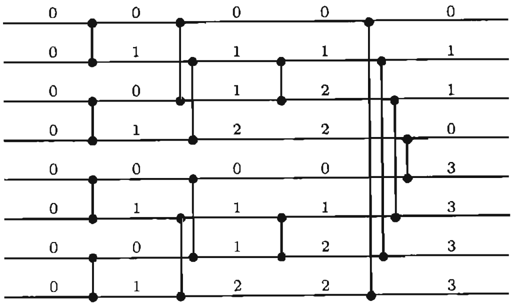


图 55　对于图 54 网络的另一种解释

置于下边的直线。得到的数$\langle m_1,m_2,\cdots,m_n\rangle$在整个网络中均有性质

$$2^{m_i} \geqslant l_i \tag{20}$$

因为这在开始时成立，而且由于(19)，通过每个比较器后它还继续成立。进而

$$m_1+m_2+\cdots+m_n$$

的最后值是在这个网络中比较器的总数，因为每个比较器都对这个和加 1。

如果这个网络选择最小的 t 个数，则诸 l_i 中有 $n-t$ 个$\geqslant t+1$；因此诸 m_i 中有 $n-t$ 个必定$\geqslant\lceil\lg(t+1)\rceil$。 ▌

当 $t=1$ 和 $t=2$ 时定理 A 的下限证明是精确的(见习题 19)。表 1 给出了对于小的 t 和 n, $\hat{U}_t(n)$, $\hat{V}_t(n)$ 和 $\hat{W}_t(n)$ 的某些值。Andrew Yao(姚期智)[Ph.D. thesis, U. of Illinois(1975)]通过证明当 $n\to\infty$ 时 $\hat{U}_3(n)=2n+\lg n+O(1)$ 和 $\hat{U}_t(n)=n\lceil\lg(t+1)\rceil+O((\log n)^{\lfloor\lg t\rfloor})$,确定了对于固定的 t, $\hat{U}_t(n)$ 的渐近特性:极小延迟时间为 $\lg n+\lfloor\lg t\rfloor\lg\lg n+O(\log\log\log n)$。N. Pippenger [*SICOMP* **20** (1991), 878~887]通过非构造性的方法已经证明,对于任何 $\epsilon>0$,每当 n 充分大(依赖于 ϵ)时,存在满足 $\hat{U}_{\lfloor n/2\rfloor}(n)\leqslant(2+\epsilon)n\lg n$ 的选择网络。

表 1　在选择网络中所需要的比较($\hat{U}_t(n)$, $\hat{V}(n)$, $\hat{W}_t(n)$)

	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$
$n=1$	(0,0,0)					
$n=2$	(1,1,1)	(0,1,1)				
$n=3$	(2,2,2)	(2,3,3)	(0,2,3)			
$n=4$	(3,3,3)	(4,5,5)	(3,5,5)	(0,3,5)		
$n=5$	(4,4,4)	(6,7,7)	(6,7,8)	(4,7,9)	(0,4,9)	
$n=6$	(5,5,5)	(8,9,9)	(8,10,10)	(8,10,12)	(5,9,12)	(0,5,12)

习　题——第一组

下列习题中的若干题深入探讨了排序网络理论,此处引入某些符号是方便的。我们令 $[i:j]$ 代表一个比较/交换模块。具有 n 个输入和 r 个比较模块的一个网络写做 $[i_1:j_1][i_2:j_2]\cdots[i_r:j_r]$,其中每个 i 和 j 都 $\leqslant n$;为了简便起见,我们称它为 *n-网络*。一个网络称做*标准的*,如果对于 $1\leqslant q\leqslant r$, $i_q<j_q$。例如,图 44 描绘了一个标准的 4-网络,以比较器序列[1:2][3:4][1:3][2:4][2:3]表示之。

正文中对于画网络图形的约定仅仅用于表示标准的网络;所有比较器 $[i:j]$ 都以从 i 到 j 的一条直线表示,其中 $i<j$。当需画出非标准网络时,可以使用从 i 到 j 的一个箭头,表示较大的数去到箭头的端点。例如,图 56 所示为对于 16 个元素的非标准网络,它的比较符是[1:2][4:3][5:6][8:7]等。习题 11 证明图 56 是一个排序网络。

如果 $x=\langle x_1,\cdots,x_n\rangle$ 是一个 n 向量, α 是一个 n 网络,则由这个网络产生的数向量 $\langle(x\alpha)_1,\cdots,(x\alpha)_n\rangle$,写做 $x\alpha$。为了简洁,令 $a\vee b=\max(a,b)$, $a\wedge b=\min(a,b)$, $\bar{a}=1-a$。于是当 $i\neq k\neq j$ 时, $(x[i:j])_i=x_i\wedge x_j$, $(x[i:j])_j=x_i\vee x_j$,以及 $(x[i:j])_k=x_k$。如果对于所有的 x 和对于 $1\leqslant i\leqslant n$, $(x\alpha)_i\leqslant(x\alpha)_{i+1}$ 时,我们说 α 是一个*排序网络*。

符号 $e^{(i)}$ 代表在位置 i 处为 1,其它处为 0 的一个向量;于是 $(e^{(i)})_j=\delta_{ij}$。符号 D_n 代表所有由 0 和 1 组成的 2^n 个 n 维向量的集合,而 P_n 代表由 $\{1,2,\cdots,n\}$ 的诸排列构成的所有 $n!$ 个向量的集合。向量 $\langle x_1\wedge y_1,\cdots,x_n\wedge y_n\rangle$ 和 $\langle x_1\vee y_1,\cdots,x_n\vee y_n\rangle$ 写成 $x\wedge y$ 和 $x\vee y$,而且如果对于所有的 i, $x_i\leqslant y_i$,则写成 $x\leqslant y$。于是 $x\leqslant y$ 当且仅当 $x\vee y=y$ 当且仅当 $x\wedge y=x$。如果 x 和 y 在

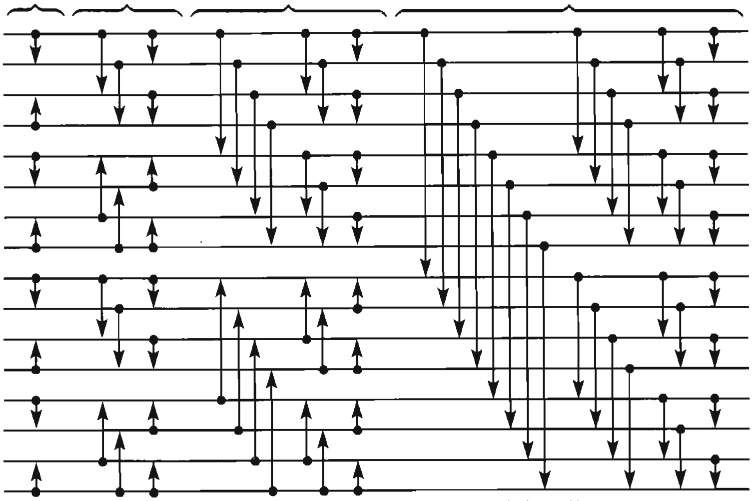


图 56 以双调排序为基础的一个非标准排序网络

D_n 中,且对某个 i,$x=(y\vee e^{(i)})\neq y$,则说 x 覆盖 y。最后,对于 D_n 中所有 x,设 $\nu(x)$ 是 x 中 1 的个数,$\zeta(x)$ 是 0 的个数;于是 $\nu(x)+\zeta(x)=n$。

1. [*20*] 当 $m=3$ 和 $n=5$ 时,试画出一个奇偶合并的网络图形。

2. [*22*] 证明 V. Pratt 的排序算法(5.2.1-30)导致对于 n 个元素的一个排序网络,它有近似于$(\log_2 n)(\log_3 n)$的延迟级。试画出对于 $n=12$ 的对应的网络。

3. [*M20*](K.E.Batcher) 试求 $C(m,m-1)$和 $C(m,m)$之间的一个简单关系。

▶4. [*M23*] 证明 $\hat{T}(6)=5$。

5. [*M16*] 证明(13)是和(10)中概述的排序网络相关联的延迟时间。

6. [*28*] 令 $T(n)$是通过进行同时的不相交的比较(不需要遵从网络限制)排序所需要的极小阶段数。每组这样的比较可表示为包含对偶集合$\{i_1:j_1,i_2:j_2,\cdots,i_r:j_r\}$的一个节点,其中 $i_1,j_1,i_2,j_2,\cdots,i_r,j_r$ 是不同的,在这个节点之下有 2^r 个分支,它们分别表示下列情况

$$\langle K_{i_1}<K_{j_1},K_{i_2}<K_{j_2},\cdots,K_{i_r}<K_{j_r}\rangle$$
$$\langle K_{i_1}>K_{j_1},K_{i_2}<K_{j_2},\cdots,K_{i_r}<K_{j_r}\rangle \quad \text{等等}$$

试证明 $T(5)=T(6)=5$。

7. [*25*] 证明如果在图 49$n=10$ 的网络中,最后的比较器恰被移到倒数第二个和倒数第三个比较器的前头,则这网络仍能完成排序。

8. [*M20*] 证明对于 $m_1,m_2,n_1,n_2\geqslant 0$,$\hat{M}(m_1+m_2,n_1+n_2)\geqslant\hat{M}(m_1,n_1)+\hat{M}(m_2,n_2)+\min(m_1,n_2)$。

9. [*M25*](R.W.Floyd) 证明 $\hat{M}(3,3)=6$,$\hat{M}(4,4)=9$,$\hat{M}(5,5)=13$。

10. [*M22*] 证明 Batcher 的双调排序器,如同在(15)之前的注释中所定义的那样,是有效的[提示:只须证明,所有的由 k 个 1,后边接以 l 个 0,后边再接以 $n-k-l$ 个 1 组成的序列都能被排序]。

11. [*M23*] 证明阶为 2^t 的 Batcher 双调排序器,将不仅对满足 $z_0\geqslant\cdots\geqslant z_k\leqslant\cdots\leqslant z_{2^t-1}$的序

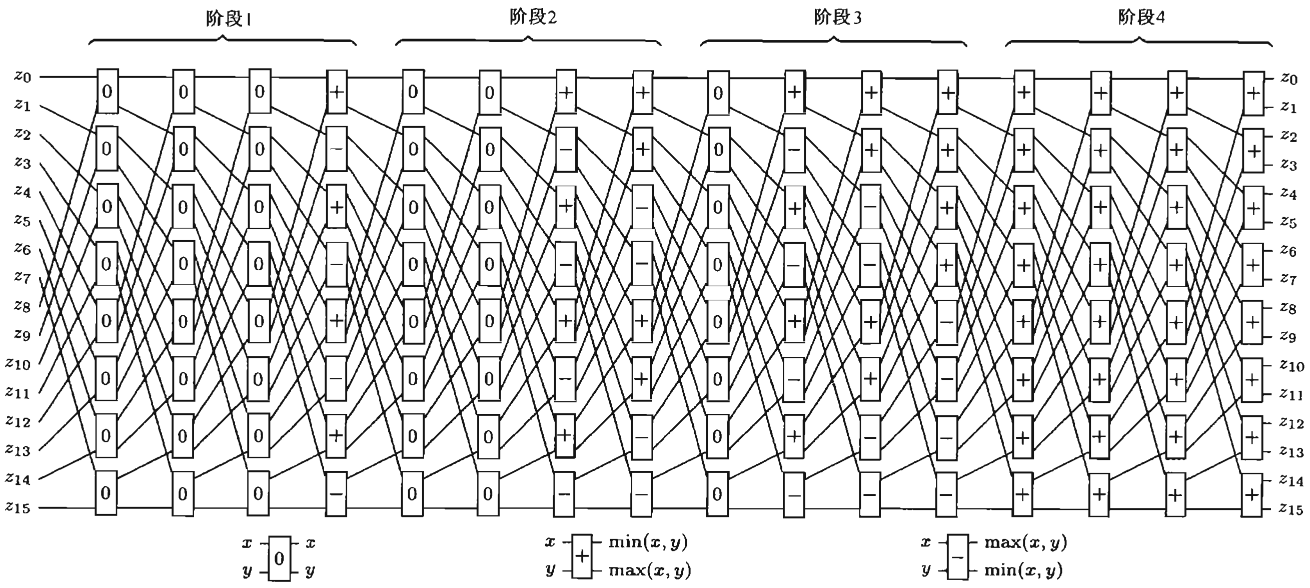


图 57 “用完全的洗牌”对 16 个元素排序

列$(z_0,z_1,\cdots,z_{2^t-1})$排序，也将对满足 $z_0\leqslant\cdots\leqslant z_k\geqslant\cdots\geqslant z_{2^t-1}$的任何序列排序[推论，图 56 中的网络将对 16 个元素排序，这是因为每个阶段由双调排序器或逆序的双调排序器组成，后者应用于已经以相反的方向排好序的序列]。

12. [*M20*] 证明或否定：如果 x 和 y 是同样长度的双调序列，则 $x\vee y$ 和 $x\wedge y$ 也是。

▶13. [*24*] (H.S.Stone) 证明对于 2^t 个元素的一个排序网络，可以仿照图 57 中对于 $t=4$ 所示的形式加以构造。在这个方案中，t^2 个步骤的每一步都由前 2^{t-1}个元素和最后 2^{t-1}个元素的一个"完全的洗牌"组成，接着是对 2^{t-1}对相邻元素执行的同时的操作。这些操作每一个或是"0"(无操作)，或是"+"(一个标准比较模块)或是"−"(反序的比较模块)。这个排序分 t 个阶段进行，每阶段又分 t 步；在最后阶段，所有的操作都是"+"。在阶段 $s(s<t)$，我们做其中所有操作都是"0"的 $t-s$ 步，后边接之以 s 步，其中在第 q 步内的操作交替地由 2^{q-1}个"+"接之以 2^{q-1}个"−"组成，$q=1,2,\cdots,s$。

[注意，这个排序方案可以通过相当简单的设备予以实施。这个设备的线路执行一个"洗牌和操作"步骤，以及把输出线反馈到输入。图 57 中的前 3 步当然可以消去；保留它们仅仅是为了使这个形式更清楚。Stone 指出，在若干其它算法中也出现有相同型式的"洗牌操作"，诸如在快速傅里叶变换中(参见 4.6.4-(40))]。

▶14. [*M27*] (V.E.Alekseev) 设 $\alpha=[i_1:j_1]\cdots[i_r:j_r]$是一个 n-网络；对于 $1\leqslant s\leqslant r$，我们定义 $\alpha^s=[i'_1:j'_1]\cdots[i'_{s-1}:j'_{s-1}][i_s:j_s]\cdots[i_r:j_r]$，其中 i'_k 和j'_k 是从 i_k 和 j_k 而来，不过当 i_s 和j_s 出现时(无论在哪)，把 i_s 改为j_s 和把j_s 改为i_s。例如，如果 $\alpha=[1:2][3:4][1:3][2:4][2:3]$，则 $\alpha^4=[1:4][3:2][1:3][2:4][2:3]$。

a)证明 $D_n\alpha=D_n(\alpha^s)$。

b)证明$(\alpha^s)^t=(\alpha^t)s$。

c)α 的一个共轭是形如$(\cdots((\alpha^{s_1})^{s_2})\cdots)^{s_k}$的任何网络，试证明 α 至多有 2^{r-1}个比较。

d)设 $g_\alpha(x)=[x\in D_n\alpha]$以及 $f_\alpha(x)=(\bar{x}_{i1}\vee x_{j1})\wedge\cdots\wedge(\bar{x}_{ir}\vee x_{jr})$。试证明 $g_\alpha(x)=\vee\{f_{\alpha'}(x)\mid\alpha'$是$\alpha$ 的一个共轭$\}$。

e)设 G_α 是具有顶点$\{1,\cdots,n\}$且对于 $1\leqslant s\leqslant\gamma$，具有有向边 $i_s\to j_s$ 的有向图。试证明 α 是一个排序网络，当且仅当对于 $1\leqslant i<n$ 和同α 共轭的所有α'，$G_{\alpha'}$有从i 到$i+1$ 的一条有向通路[这个条件是稍微突出的，因为 G_α 不依赖于α 中比较器的阶]。

15. [*20*] 试求 4 个元素的一个非标准排序网络，它只有 5 个比较模块。

16. [*M22*] 证明：下列算法把任何一个排序网络$[i_1:j_1],\cdots,[i_r:j_r]$都变换成相同长度的一个标准的排序网络。

T1. 令 q 是使得$i_q>j_q$ 的最小下标，如果这样的下标不存在，就停止。

T2. 对于 $q\leqslant s\leqslant r$，在所有比较器$[i_s:j_s]$中把 i_q 的所有出现改为j_q，并把 j_q 的所有出现改为i_q。返回 T1。 ▎

因此，网络[4:1][3:2][1:3][2:4][1:2][3:4]首先变换成[1:4][3:2][4:3][2:1][4:2][3:1]，然后[1:4][2:3][4:2][3:1][4:3][2:1]，然后[1:4][2:3][2:4][3:1][2:3][4:1]等等，直到得到标准网络[1:4][2:3][2:4][1:3][1:2][3:4]为止。

17. [*M25*] 令 D_{tn}是恰有 t 个 1 的所有$\binom{n}{t}$个 0 和 1 的序列$\langle x_1,\cdots,x_n\rangle$的集合。证明：$\hat{U}_t(n)$是在对 D_{tn}的所有元素排序的一个网络中所需要的比较器的极小个数；$\hat{V}_t(n)$是为排序 $D_{tn}\cup D_{(t-1)n}$所需要的比较器的极小个数；而 $\hat{W}_t(n)$是为对$\bigcup_{0\leqslant k\leqslant t}D_{kn}$排序所需要的比较器的极小个数。

▶18. [*M20*] 证明求 $2t-1$ 个元素的中值的一个网络至少需要 $(t-1)\lceil\lg(t+1)\rceil+\lceil\lg t\rceil$ 个比较模块[提示:见定理 A 的证明]。

19. [*M22*] 证明:对所有 $n\geqslant 2$, $\hat{U}_2(n)=2n-4$ 和 $\hat{V}_2(n)=2n-3$。

20. [*24*] 证明 $\hat{V}_3(5)=7$。

21. [*21*] 真或假:插入一个新的标准比较器到任何标准排序网络中将产生另一个标准的排序网络。

22. [*M17*] 设 α 是任何 n-网络,并设 x 和 y 是 n-向量。

a)证明 $x\leqslant y$ 意味着 $x\alpha\leqslant y\alpha$。

b)证明 $x\cdot y\leqslant(x\alpha)\cdot(y\alpha)$,其中 $x\cdot y$ 表示点积 $x_1y_1+\cdots+x_ny_n$。

23. [*M18*] 设 α 是一个 n-网络,证明有一个排列 $p\in P_n$ 使得 $(p\alpha)_i=j$,当且仅当在 D_n 中存在向量 x 和 y 使得 x 覆盖 y, $(x\alpha)_i=1$, $(y\alpha)_i=0$ 和 $\zeta(y)=j$。

▶24. [*M21*] (B. E. Alekseev) 设 α 是一个 n-网络,而且对于 $1\leqslant k\leqslant n$,令 $l_k=\min\{(p\alpha)_k\mid p\in P_n\}$, $U_k=\max\{(p\alpha)_k\mid p\in P_n\}$,表示可出现于输出线 k 中值的下限和上限。对于网络 $\alpha'=\alpha[i:j]$类似地定义 l'_k 和 u'_k。证明

$$l'_i=l_i\wedge l_j,\ l'_j\leqslant l_i+u_j,\ u'_i\geqslant u_i+u_j-(n+1),\ u'_j=u_i\vee u_j$$

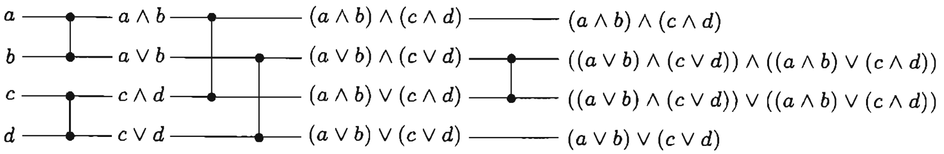
[提示:给定 D_n 中的向量 x 和 y,有 $(x\alpha)_i=(y\alpha)_j=0$, $\zeta(x)=l_i$, $\zeta(y)=l_j$,求出 D_n 中的一个向量 z,使 $(z\alpha')_j=0$, $\zeta(z)\leqslant l_i+l_j$。]

25. [*M30*] 设 l_k 和 u_k 如习题 24 中所定义的那样。证明集合 $\{(p\alpha)_k\mid p$ 在 P_n 中$\}$包括 l_k 和 u_k 之间以及 l_k 和 u_k 本身的所有整数。

26. [*M24*] (R. W. Floyd) 设 α 是一个 n-网络。证明集合 $D_n\alpha=\{x\alpha\mid x$ 在 D_n 中$\}$可以由集合 $P_n\alpha=\{p\alpha\mid p$ 在 P_n 中$\}$确定;反之, $P_n\alpha$ 可由 $D_n\alpha$ 确定。

▶27. [*M20*] 设 x 和 y 是向量,并设 $x\alpha$ 和 $y\alpha$ 被排序,证明 $(x\alpha)_i\leqslant(y\alpha)_j$,当且仅当每从 y 中任取 j 个元素,我们都可以从 x 中选择 i 个元素,使得每个选定的 x 元素$\leqslant$某个选定的 y 元素。利用这一原理证明,*如果对任何矩阵先按行,后按列排序,则诸行仍保持有序*。

▶28. [*M20*] 下列图形说明,有可能利用输入元素系统地写出在一个排序网络中所有直线上的内容的公式:



利用交换律 $x\wedge y=y\wedge x$, $x\vee y=y\vee x$,结合律 $x\wedge(y\wedge z)=(x\wedge y)\wedge z$, $x\vee(y\vee z)=(x\vee y)\vee z$,分配律 $x\wedge(y\vee z)=(x\wedge y)\vee(x\wedge z)$, $x\vee(y\wedge z)=(x\vee y)\wedge(x\vee z)$吸收律 $x\wedge(x\vee y)=x\vee(x\wedge y)=x$ 以及等幂律 $x\wedge x=x\vee x=x$,可以把这个网络右端的公式分别简化为 $(a\wedge b\wedge c\wedge d)$, $(a\wedge b\wedge c)\vee(a\wedge b\wedge d)\vee(a\wedge c\wedge d)\vee(b\wedge c\wedge d)$, $(a\wedge b)\vee(a\wedge c)\vee(a\wedge d)\vee(b\wedge c)\vee(b\wedge d)\vee(c\wedge d)$, $a\vee b\vee c\vee d$。

证明,一般地, $\{x_1,\cdots,x_n\}$的第 t 个最大元素由"初等对称函数"

$\sigma_t(x_1,\cdots,x_n)=\vee\{x_{i_1}\wedge x_{i_2}\wedge\cdots\wedge x_{i_t}\mid 1\leqslant i_1<i_2<\cdots<i_t\leqslant n\}$给出[有 $\binom{n}{t}$ 项被 $\vee$ 在一起。

于是求极小花费的排序网络的问题，等价于用极小数量的“与/或”线路计算初等对称函数的问题，其中在每级上我们用 $\phi\wedge\psi$ 和 $\phi\vee\psi$ 代替两个量 ϕ 和 ψ]。

29. [*M20*] 设 $x_1\leqslant x_2\leqslant x_3$ 和 $y_1\leqslant y_2\leqslant y_3\leqslant y_4\leqslant y_5$，又设 $z_1\leqslant z_2\leqslant\cdots\leqslant z_8$ 是把诸 x 和诸 y 合并的结果，应用运算符 $\wedge$ 和 $\vee$，求出以 x 和 y 表达的每一个 z 的公式。

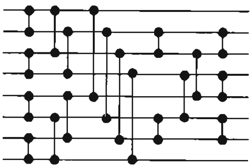
30. [*HM24*] 证明：使用习题 28 中的恒等式，可把任何包含 $\wedge$ 和 $\vee$，以及独立变量 $\{x_1,\cdots,x_n\}$ 的公式，归结为一个“正则”的形式 $\tau_1\vee\tau_2\cdots\vee\tau_k$，其中 $k\geqslant 1$，每个 τ_i 形为 $\wedge\{x_j\mid j$ 在 S_i 中$\}$，其中 S_i 是 $\{1,2,\cdots,n\}$ 的一个子集，而且当 $i\neq j$ 时，集合 S_i 不包括在 S_j 中。证明两个这样的正则形式对于所有的 $x_1,\cdots,x_n$ 都相等的充要条件是它们恒等(准确到次序)。

31. [*M24*] (R. Dedekind, 1897) 设 δ_n 是在习题 30 的意义下关于 $x_1,\cdots,x_n$ 的不同正则形式的个数。于是 $\delta_1=1$，$\delta_2=4$ 和 $\delta_3=18$，问 δ_4 是多少？

32. [*M28*] (M. W. Green) 设 $G_1=\{00,01,11\}$，G_{t+1}是所有串 $\theta\phi\psi\omega$ 的集合，使得 $\theta,\varphi,\psi,\omega$ 的长度为 2^{n-1}且 $\theta\phi,\psi\omega,\theta\psi$ 及 $\phi\omega$ 在 G_n 中。设 α 是由图 49 所示 16 排序器的前四级组成的网络。证明 $D_{16}\alpha=G_4$，并且证明，它恰有 δ_4+2 个元素(参见习题 31)。

▶33. [*M22*] 习题 31 中，并非$\langle x_1,\cdots,x_n\rangle$的函数的所有 δ_n 都可以出现于比较器网络中。证明，函数$(x_1\wedge x_2)\vee(x_2\wedge x_3)\vee(x_3\wedge x_4)$事实上不可能成为关于$\langle x_1,\cdots,x_n\rangle$的任何比较器网络的输出。

34. [*23*] 下图是否是一个排序网络？



35. [*20*] 证明对于 $1\leqslant i<n$，任何标准排序网络都必须至少包含每个相邻的比较器$[i:i+1]$一次。

▶36. [*22*] 图 47 的网络仅包含相邻的比较$[i:i+1]$；我们称这个网络为*本原的*。

a)证明 n 个元素的一个本原排序网络至少有$\binom{n}{2}$个比较器[提示：考虑一个排列的逆]。

b)(R. W. Floyd, 1964) 设 α 是 n 个元素的一个本原网络，又设 x 是对于某个 $i<j$ 使得$(x\alpha)_i>(x\alpha)_j$ 的一个向量。证明$(y\alpha)_i>(y\alpha)_j$，其中 y 是向量$\langle n,n-1,\cdots,1\rangle$。

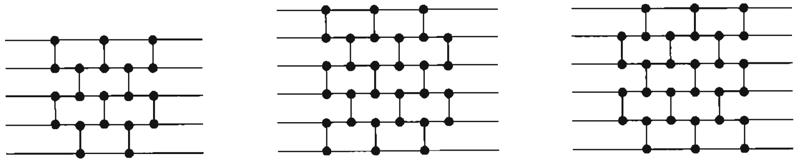
c)作为 b)的推论，一个本原网络是一个排序网络的充要条件是，它把单个向量$\langle n,n-1,\cdots,1\rangle$排序。

37. [*M22*] 对于 $n\geqslant 3$ 的 n 个数的*奇偶转置排序*，是如图 58 所示的，一个排成类似砖块式的，具有$\frac{1}{2}n(n-1)$个比较器的，深度为 n 层的网络(当 n 是偶数时有两种可能性)。这种排序在硬件中是特别容易实现的，因为仅仅交替地执行两类操作。证明，这样一个网络是一个有效的排序网络[提示：见习题 36]。

▶38. [*43*] 设 $N=\binom{n}{2}$，试求在形如$(n-1,n-2,\cdots,1)$的杨氏表图和本原排序网络$[i_1:i_1+1]\cdots[i_N:i_{N+1}]$之间的一一对应[由定理 5.1.4H，恰好有

$$\frac{N!}{1^{n-1}3^{n-2}5^{n-3}\cdots(2n-3)^1}$$

个这样的网络]。提示：习题 36(c)表明，没有多余比较器的本原网络，对应于 5.1.1 小节中类似图 1 的多面体中从 1 2 … n 到 n … 2 1 的通路。



$n=5$　　　　$n=6$　　　　$n=6$

图 58　奇偶转置排序

39. [*25*] 假设已知 n 条线上的本原比较器网络正确地对单个输入 1010…10 进行排序(参见习题 36,假定 n 为偶数)。证明它的“中间第三个”(由线$\lceil n/3\rceil$到$\lceil 2n/3\rceil$(含两者)的所有比较器组成),将对所有输入进行排序。

40. [*HM44*] 随机地选择比较器$[i_1:i_1+1][i_2:i_2+1]\cdots[i_r:i_r+1]$，而且 $i_k\in\{1,2,\cdots,n-1\}$的每个值都同等可能；当这个网络包含类似图 47 的一个冒泡排序格局作为一个子网络时，这个过程停止。试证明，除了有 $O(n^{-1000})$的概率之外，$r\leqslant 4n^2+\sqrt{n}\lg n$。

41. [*M47*] 随机地选择比较器$[i_1:j_1][i_2:j_2]\cdots[i_r:j_r]$，而且每个非冗余的选择 $1\leqslant i_k<j_k\leqslant n$ 同等可能；当已经得到一个排序网络时，这个过程停止。试估计 r 的预期的值；对于所有 $\epsilon>0$，它是 $O(n^{1+\epsilon})$吗?

▶42. [*25*] (D. Van Voorhis) 证明：$\hat{S}(n)\geqslant\hat{S}(n-1)+\lceil\lg n\rceil$。

43. [*48*] 试求具有少于 $C(m,n)$个比较器的一个(m,n)合并网络，或者证明它不存在。

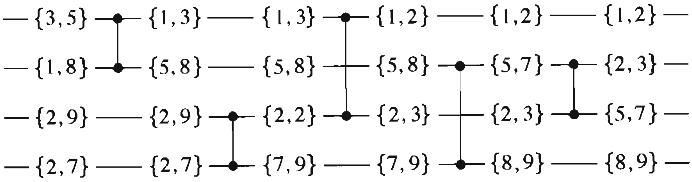
44. [*50*] 对于某些 $n>8$，求 $\hat{S}(n)$的精确值。

45. [*M20*] 试证明，没有多个扇区输出的任何一个$(1,n)$合并网络，至少有$\lceil\lg(n+1)\rceil$层延迟。

46. [*30*] (M. Aigner) 证明，使用如同习题 6 中那样的，进行同时不相交比较的任何算法，为把 m 个元素同 n 个元素合并，所需要的极小阶段数至少是$\lceil\lg(m+n)\rceil$；因此双调合并网络有最优延迟。

47. [*47*] 对于某个 n，习题 6 的函数 $T(n)$是否严格地小于 $\hat{T}(n)$?

▶48. [*26*] 我们可以用另一种方式来解释排序网络，让每条线载有 m 个数的一个多重集合，而不是单个数；在这种解释之下，操作$[i:j]$分别以 $x_i\curlywedge x_j$ 和 $x_i\curlyvee x_j$ 代替 x_i 和 x_j，即 $2m$ 个数 $x_i\uplus x_j$ 中最小的 m 个和最大的 m 个(例如，图式



图解了当 $m=2$ 时的这个解释；每个比较器合并它的输入并把低的一半和高的一半分开)。

如果 a 和 b 每一个都是 m 个数的多重集体，我们说 $a\ll b$ 当且仅当 $a\curlywedge b=a$(等价地 $a\curlyvee b=b$，a 的最大元素小于或等于 b 的最小)。于是 $a\curlywedge b\ll a\curlyvee b$。

设 α 是一个 n-网络，又设 $x=\langle x_1,\cdots,x_n\rangle$ 是一个向量，其中每个 x_i 是 m 个元素的一个多重集合。证明，在上述解释中，如果 $(x\alpha)_i$ 不是 $\ll(x\alpha)_j$，则在 D_n 中有一个向量 y，使得 $(y\alpha)_i=1$ 和 $(y\alpha)_j=0$[因此，如果我们以 m 路合并来代替比较，则 n 个元素的一个排序网络变成 mn 个元素的一个排序网络。图 59 所示为使用这种办法由 4 元排序器构造的 8 元排序器]。

▶49. [*M23*] 使用习题 48 中的符号标记，证明 $(x ⩓ y) ⩓ z = x ⩓ (y ⩓ z)$ 和 $(x ⩔ y) ⩔ z = x ⩔ (y ⩔ z)$；然而 $(x ⩔ y) ⩓ z$ 不总等于 $(x ⩓ z) ⩓ (y ⩓ z)$，而且 $(x ⩓ y) ⩔ (x ⩓ z) ⩔ (y ⩓ z)$ 不总等于 $x \uplus y \uplus z$ 中间的 m 个元素。对于这些中间元素，试借助于 x,y,z 以及 $⩓$ 和 $⩔$ 运算，求出一个正确的公式。

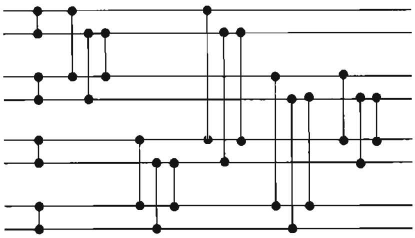


图 59 通过使用合并解释，从一个 4-排序器构造一个 8-排序器

50. [*HM46*] 考察习题 48 中定义的 $⩓$ 和 $⩔$ 操作的性质，是否有可能以某种漂亮的方式来表征这个代数中的所有恒等式，或者从有限的恒等式的集合来推导出所有的恒等式来？在这方面，诸如 $x ⩓ x ⩓ x = x ⩓ x$，或者 $x ⩓ (x ⩔ (x ⩓ (x ⩔ y))) = x ⩓ (x ⩔ y)$ 这样的恒等式，仅对 $m\leqslant 2$ 成立，有较小的兴趣；仅仅考虑对于所有的 m 成立的恒等式。

▶51. [*M25*] (R.L.Graham) 我们说比较器 $[i:j]$ 在网络 $\alpha_1[i,j]\alpha_2$ 中是多余的，如果对于所有向量 x，$(x\alpha_1)_i\leqslant(x\alpha_1)_j$，或对所有向量 x，$(x\alpha_1)_i\geqslant(x\alpha_1)_j$。证明，如果 α 是具有 r 个非多余比较器的网络，则至少有不同下标的 r 个不同的有序对 (i,j)，使得对于所有向量 x，$(x\alpha)_i\leqslant(x\alpha)_j$（因此，不带多余比较器的一个网络至多包含 $\binom{n}{2}$ 个模块）。

▶52. [*32*] (M.O.Rabin,1980) 试证明，通过考虑图 61 所勾勒的形式的一个网络，来一般地判定一系列的比较器是否定义一个排序网络，是固有地困难的。把输入编号为 x_0 到 x_N 是方便的，其中 $N=2mn+m+2n$；正整数 m 和 n 是参数。对于 $1\leqslant j\leqslant 2n$ 和 $1\leqslant k\leqslant m$，第一个比较器

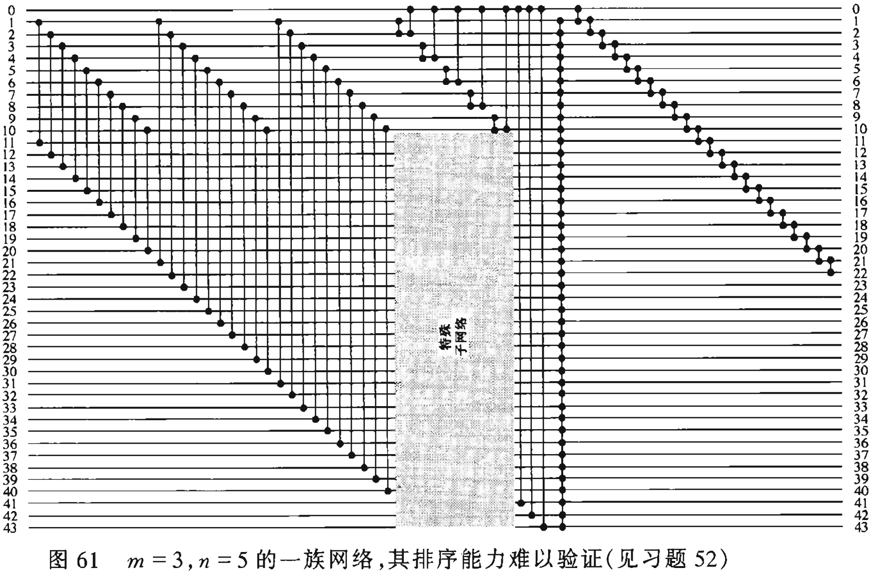


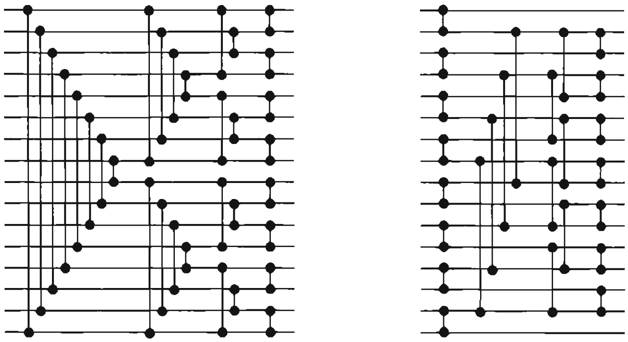
图 61 $m=3,n=5$ 的一族网络，其排序能力难以验证(见习题 52)

为$[j:j+2nk]$。然后对于$1\leqslant j\leqslant n$，我们有$[2j-1:2j][0:2j]$，并行于只使用$>2n$的下标的一个特殊子网络。其次，对于$1\leqslant j\leqslant m$，我们比较$[0:2mn+2n+j]$。最后对于$\langle x_1,\cdots,x_n\rangle$，有一个完备的排序网络，其后为有$[0:1][1:2]\cdots[N-t-1:N-t]$，其中$t=mn+n+1$。

a)借助于特殊子网络的特性，描述由这样的一个网络排不了序的所有输入$\langle x_0,x_1,\cdots,x_N\rangle$。

b)给定如同$(y_1\vee y_2\vee \bar{y}_3)\wedge(\bar{y}_2\vee y_3\vee \bar{y}_4)\wedge\cdots$这样的一组短语，说明怎样来构造一个特殊的子网络，使得图 61 对所有输入排序当且仅当这些短语是不可满足的[因此在 7.9 节的意义下，判定一个比较器序列是否形成一个排序网络是一个协 NP 完全的]。

53. [*30*] (*周期的排序网络*) 下列两个 16-网络图解在$t=4$的情况下，对于$n=2^t$的t级网络的一般递归构造：

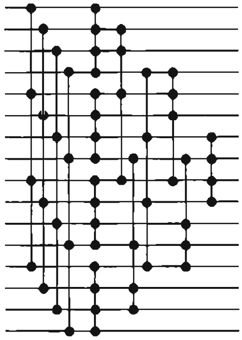


(a)　　　　　　(b)

如果我们从 0 到2^t-1对输入线进行编号，则在情况(a)中的第l级有比较器$[i:j]$，其中$i \bmod 2^{t+1-l}<2^{t-l}$和$j=i\oplus(2^{t+1-l}-1)$；如同在双调合并中那样，总共有$t2^{t-1}$个比较器。在情况(b)中，对于$0\leqslant j<2^{t-1}$第一级比较器为$[2j:2j+1]$；当$2\leqslant l\leqslant t$，对于$0\leqslant j<2^{t-1}-2^{t-l}$，第$l$级比较器为$[2j+1:2j+2^{t+1-l}]$；如同在奇偶合并中那样，总共有$(t-1)2^{t-1}+1$个比较器。

如果对于某个$k\geqslant 1$，在定理 5.2.1H 的意义下，输入的个数是2^k-阶的，试证明两个网络产生的输出均为2^{k-1}阶。因此，我们可以通过把它们传送到两个网络的任一个t次，来对2^t个数进行排序[当t很大时，这些排序网络大约使用两倍于算法 5.2.2M 的比较；但总共的延迟时间和在图 57 中相同，而且实现更简单，因为重复地使用相同的网络]。

54. [*42*] 试分析由m排序器模块，而不是 2 排序器模块，组成的排序网络的性质(例如，G. Shapiro 已经构造了如下网络



它使用 14 个 4 分类器来对 16 个元素进行排序,这是最好的可能吗? 试证明,当 m 充分大时,m^2 个元素可以通过最多 16 级的 m 排序器来加以排序)。

55. [*23*] 一个排列网络是一个模块序列 $[i_1:j_1]\cdots[i_r:j_r]$,其中每个模块 $[i:j]$ 可以通过外部控制来设置,是将输入原样输出,还是将 x_i 和 x_j 交换(不管 x_i 和 x_j 的值是什么),并且每个输入的排列可以通过模块的某种设定,在输出线上得以实现。每个排序网络显然是一个排列网络,但是反之则不成立:试找出 5 个元素的一个排列网络,它仅有 8 个模块。

▶56. [*25*] 假设二进位向量 $x\in D_n$ 未被排序。试证明,存在一个标准的 n-网络 α_x,尽管它能对 D_n 的所有其它元素进行排序,但它不能对 α 进行排序。

57. [*M35*] 偶奇合并类似于 Batcher 的奇偶合并,只是当 $mn>2$ 时在做类似于(1)的一组 $\lceil m/2\rceil+\lceil n/2\rceil-1$ 个比较-交换之前,它递归地把序列 $\langle x_{m \bmod 2+1},\cdots,x_{m-3},x_{m-1}\rangle$ 同 $\langle y_1,y_3,\cdots,y_{2\lceil n/2\rceil-1}\rangle$,以及把 $\langle x_{(m+1)\bmod 2+1},\cdots,x_{m-2},x_m\rangle$ 同 $\langle y_2,y_4,\cdots,y_{2\lfloor n/2\rfloor}\rangle$ 合并。试证明,无须做多于双调方法的比较,偶奇合并即可实现双调合并的最优延迟时间 $\lceil\lg(m+n)\rceil$。实际上,即证明由偶奇合并所做的比较次数 $A(m,n)$,满足 $C(m,n)\leqslant A(m,n)<\frac{1}{2}(m+n)\lg\min(m,n)+m+\frac{3}{2}n$。

习 题——第二组

下列习题涉及了同排序有关的若干不同类型的最优性问题。前面的少数问题,是以 P. N. Armstrong 和 R. J. Nelson 早在 1954 年即研究过的,冒泡排序有趣的"多头"推广为基础的[见*U. S. Patents* 3029413,3034102]。设 $1=h_1<h_2<\cdots<h_m=n$ 是一个递增的整数序列;我们称它做长度为 m 和间距为 n 的一个"头序列",并用它来定义一种特殊类型的排序方法。记录 $R_1,\cdots,R_N$ 的排序分若干次扫描进行,每次扫描由 $N+n-1$ 个步骤组成。在第 j 步上($j=1-n,2-n,\cdots,N-1$),应考察并在必要时重新排列记录 $R_{j+h[1]},R_{j+h[2]},\cdots,R_{j+h[m]}$,使得它们的键码成为有序的(这时我们说 $R_{j+h[1]},\cdots,R_{j+h[m]}$ 是"在读写头之下"。当 $j+h[k]<1$ 或 $>N$ 时,不考虑记录 $R_{j+h[k]}$;从效果上看键码 $K_0,K_{-1},K_{-2},\cdots$ 被处理做 $-\infty$,而 $K_{N+1},K_{N+2},\cdots$ 被处理做 $+\infty$。因此当 $j\leqslant-h[m-1]$ 或 $j>N-h[2]$ 时,步骤 j 实际上是显然的)。

例如,下表示出了当 $m=3,N=9$ 以及 $h_1=1,h_2=2,h_3=4$ 时一个排序的一次扫描:

	K_{-2}	K_{-1}	K_0	K_1	K_2	K_3	K_4	K_5	K_6	K_7	K_8	K_9	K_{10}	K_{11}	K_{12}
$j=-3$	__	__		$\underline{3}$	1	4	5	9	2	6	8	7			
$j=-2$		__	__	3	$\underline{1}$	4	5	9	2	6	8	7			
$j=-1$			__	$\underline{3}$	1	$\underline{4}$	5	9	2	6	8	7			
$j=0$				$\underline{1}$	$\underline{3}$	4	$\underline{5}$	9	2	6	8	7			
$j=1$				1	$\underline{3}$	$\underline{4}$	5	$\underline{9}$	2	6	8	7			
$j=2$				1	3	$\underline{2}$	$\underline{4}$	9	$\underline{5}$	6	8	7			
$j=3$				1	3	2	$\underline{4}$	$\underline{6}$	5	$\underline{9}$	8	7			
$j=4$				1	3	2	4	$\underline{5}$	$\underline{6}$	9	$\underline{8}$	7			
$j=5$				1	3	2	4	5	$\underline{6}$	$\underline{7}$	8	$\underline{9}$			

$j=6$	1	3	2	4	5	6	$\underline{7}$	$\underline{8}$	9	_	
$j=7$	1	3	2	4	5	6	7	$\underline{8}$	$\underline{9}$		_
$j=8$	1	3	2	4	5	6	7	8	$\underline{9}$	_	_

当 $m=2,h_1=1$ 及 $h_2=2$ 时,这个多头的方法简化成为冒泡排序(算法 5.2.2B)。

58. [*21*] (James Dugundji) 证明如果对于某个 $k,1\leqslant k\leqslant m$,有 $h[k+1]=h[k]+1$,则上边定义的多头排序器将在有限次扫描中,最终完成对任何输入文件的排序。但是如果对于 $1\leqslant k<m,h[k+1]\geqslant h[k]+2$,则这个输入可能永远排不好序。

▶59. [*30*] (Armstrong 和 Nelson) 设对 $1\leqslant k<m,h[k+1]\leqslant h[k]+k$,又设 $N\geqslant n-1$,证明经第一次扫描后最大的 $n-1$ 个元素总是被送到它们最终目的地[提示:使用 0—1 原理;如果对一些 0 和 1 进行排序,且 1 的个数少于 n,证明不可能所有的头都读出 1,除非所有的 0 都在这些头的左边]。

证明当这些头满足给定条件时,排序将在至多$\lceil (N-1)/(n-1)\rceil$次扫描中完成。是否有一个输入文件,它需要这么多遍扫描?

60. [*26*] 如果 $n=N$,证明:当且仅当对于 $1\leqslant k<m,h[k+1]\leqslant 2h[k]$时,第一次扫描可以保证把最小的键码放置到位置 R_1 上。

61. [*34*] (J. Hopcroft) N 个元素的一个"完全排序器",是 $N=n$ 的一个总在一次扫描中完成的多头排序器。习题 59 证明,序列$\langle h_1,h_2,h_3,h_4,\cdots,h_m\rangle=\left\langle 1,2,4,7,\cdots,1+\binom{m}{2}\right\rangle$给出一个完全排序器,该排序器对于 $N=\binom{m}{2}+1$ 个元素,使用 $m=(\sqrt{8N-7}+1)/2$ 个头。例如,头序列$\langle 1,2,4,7,11,16,22\rangle$是对于 22 个元素的完全排序器。

证明,事实上头序列$\langle 1,2,4,7,11,16,23\rangle$是对于 23 个元素的一个完全排序器。

62. [*49*] 给定 m,分析使 m 头完全排序器存在的最大N。$N=O(m^2)$吗?

63. [*23*] (V. Pratt) 若对 $1\leqslant k\leqslant m$,每个头 h_k 都在位置 2^{k-1}上,现要对 0 和 1 的序列 $z_1,z_2\cdots z_{2^m-1}$进行排序,其中当且仅当 j 是 2 的一个乘方时 $z_j=0$,问需要多少次扫描?

64. [*24*] (一致排序) 5.3.1 小节中图 34 的树在第一级的两个分支中进行了 2:3 的比较,而在第二级的每个分支中比较 1:3(除非该比较是多余的)。一般地说,我们可以考虑在这种意义下一致的所有排序算法类;假定 $M=\binom{N}{2}$个对偶$\{(a,b)\mid 1\leqslant a<b\leqslant N\}$已经安排成一个序列

$$(a_1,b_1),(a_2,b_2),\cdots,(a_M,b_M)$$

我们可以逐个地对未知结果的对进行比较 $K_{a_1}:K_{b_1},K_{a_2}:K_{b_2},\cdots$诸对偶$(a,b)$共有 $M!$ 个排列,其中每个都定义了一个一致的算法。一致排序的概念是 H. L. Beus 给出的[*JACM* **17**(1970),482～495],他的工作为下面的几个习题给出了提示。

借助于图论来正式地定义一致排序是方便的。设 G 是顶点$\{1,2,\cdots,N\}$上的有向图,没有有向边,对于 $i=1,2,\cdots,M$,我们对 G 加上有向边如下:

情况 1 G 包含从 a_i 到 b_i 的一条路径。把有向边 $a_i\rightarrow b_i$ 加到 G 中。

情况 2 G 包含从 b_i 到 a_i 的一条路径。把有向边 $b_i\rightarrow a_i$ 加到 G 中。

情况 3 G 不包含从 a_i 到 b_i 或 b_i 到 a_i 的路径。比较 $K_{a_i}:K_{b_i}$,如果 $K_{a_i}\leqslant K_{b_i}$,则把有向边 $a_i\rightarrow b_i$ 加到 G 中;如果 $K_{a_i}>K_{b_i}$,则把有向边 $b_i\rightarrow a_i$ 加到 G 中。

我们主要关心由一个一致排序算法所做的键码比较次数，而不是关心用什么方法避免多余的比较；图 G 不必明显地构造，这里使用它仅仅是用它来帮助定义一致排序的概念。

我们也将考虑有限制的一致排序，其中在上述的情况 1、2 和 3 中，仅仅计算长度为 2 的路径（一个有限制的一致排序算法可能会进行某些多余的比较，但习题 65 说明在有限制的情况下分析要更简单些）。

证明当对偶序列按字典次序排列

$$(1,2)(1,3)(1,4)\cdots(1,N)(2,3)(2,4)\cdots(2,N)\cdots(N-1,N)$$

时，有限制的一致算法和一致算法是一样的。证明，事实上，当键码不同且快速排序的多余比较如习题 5.2.2-24 中那样被消除时（忽略在快速排序中实际进行的比较次序，仅仅考虑哪些键码对被比较过），这两个算法都等价于"快速排序"（算法 5.2.2Q）。

65. [*M38*] 如同在习题 64 中那样，给定一个对偶序列 $(a_1,b_1)\cdots(a_M,b_M)$，设 c_i 是使得 $j<k<i$ 且使 $(a_i,b_i),(a_j,b_j),(a_k,b_k)$ 形成一个三角形的对偶 (j,k) 的数目。

a) 证明通过有限制的一致排序算法所做的平均比较次数为 $\sum_{i=1}^{M} 2/(c_i+2)$。

b) 使用 a) 和习题 64 的结果来确定快速排序执行的非多余比较的平均次数。

c) 合并排序引起了（但不等价于）下列的对偶序列：

$(1,2)(3,4)(5,6)\cdots(1,3)(1,4)(2,3)(2,4)(5,7)\cdots(1,5)(1,6)(1,7)(1,8)(2,5)\cdots$

试问以这个序列为基础的一致方法，平均说来，比快速排序执行更多还是更少的比较？

66. [*M29*] 在最坏的情况下，快速排序进行 $\binom{N}{2}$ 次比较。是否所有有限制的一致排序算法（在习题 63 的意义下）在其最坏的情况下，都执行 $\binom{N}{2}$ 次比较？

67. [*M48*] (H. L. Beus) 就所有（有限制的）一致排序算法而言，快速排序是否有极小的平均比较次数？

68. [*25*] Howard B. Demuth 的博士论文"Electronic Data Sorting"(Stanford University, October 1956) 也许是详细讨论计算复杂性问题的第一篇论文。Demuth 考虑了排序设备的若干抽象模型，并建立了每个模型可达到的平均和极大执行时间的下限和上限。他的最简单的模型"循环无逆存储"（见图 60），是本习题的主题。

考虑一台机器，它在若干次扫描中对 $R_1R_2\cdots R_N$ 排序，其中每次扫描都包含下列 $N+1$ 步：

步骤 1 置 $R\leftarrow R_1$（R 是机器内部的一个寄存器）。

步骤 i 对于 $1<i\leq N$，或者是 (i) 置 $R_{i-1}\leftarrow R$，$R\leftarrow R_i$；或者是 (ii) 置 $R_{i-1}\leftarrow R_i$，R 保持不变。

步骤 $N+1$ 置 $R_N\leftarrow R$。

问题是找出一种方式，每次在方案 (i) 和 (ii) 中进行选择，以便把进行排序所需要的扫描次数极小化。

证明对这个模型来说，"冒泡排序"技术是最优的。换句话说，证明每当 $R\leq R_i$ 时选择方案 (i)，以及每当 $R>R_i$ 时选择方案 (ii) 的这个策略，将达到极小的扫描次数。

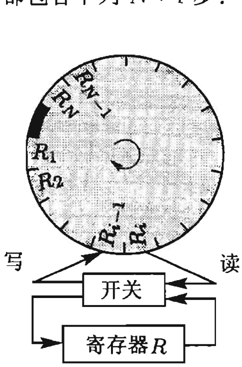


图 60 一种设备，对它来说冒泡排序是最优的

They that weave networks shall be confounded.

——Isaiah 19:9

5.4 外部排序

现在该是我们来研究当有待排序的记录数大于计算机的高速内部存储器所能容纳的数量时,出现的一些有趣问题的时候了。外部排序与内部排序十分不同,因为对外部文件进行存取的有效技术受到相当严格的限制,尽管在两种情况下排序问题都是把一个给定文件排成非减的次序。对外部排序来说,必须把数据结构安排成使得相当慢的外部存储设备(磁带、磁盘、磁鼓等),能快速地满足排序算法的要求。因而,我们已经研究过的大多数内部排序技术(插入,合并,选择),实际上对于外部排序是无用的,从而有必要重新考虑全部的问题。

例如,假设要对一个有 500 万个记录 $R_1R_2\cdots R_{5000000}$ 的文件排序,而且每个记录 R_i 是 20 个字长的(尽管键码 K_i 不必这么长)。如果这些记录中一次只能有 100 万个装入一台计算机的内部存储器中,则应怎么做呢?

一个相当明显的解决方法是,先独立地对 5 个子文件 $R_1\cdots R_{1000000}$, $R_{1000001}\cdots R_{2000000}$, $\cdots$, $R_{4000001}\cdots R_{5000000}$ 排序,然后把得到的子文件合并在一起。幸而,这个合并过程仅仅使用非常简单的数据结构,即以与栈或队列相同的顺序方式被访问的线性表;因此合并可以毫无困难地在最少费用的外部存储设备上完成。

刚才描述的过程——先内部排序,再外部合并——的使用是非常普遍的,我们将把对于外部排序的研究主要集中在这个主题的各种变形上。

通过初始的内部排序阶段产生的记录的递增序列,在关于排序的公开著作中通常称做*串*;这个术语是相当广泛的,可惜的是,它同计算机科学的其它分支中另一种更广泛的用法相抵触,在这种更广泛的用法下,串是字符的任意序列。在关于排列的研究中,已经给一个文件的排好序的段落取了一个好名字,即习惯上它被称做递增的路段或简单地说是*路段*,因此,我们将继续使用"路段"这个词来描述一个文件的已排好序的部分。用这种方法,就可能把"路段的串"和"串的路段"两者区别开来而不致引起混淆。

现在首先考虑使用磁带作为辅助存储时的外部排序过程。遵循算法 5.2.4N,S 和 L 的中心思想,用磁带进行合并的最简单和具有吸引力的方法也许是*平衡的两路合并*。在这个过程中,我们使用 4 个"工作带"。在第一阶段,由内部排序产生的递增路段被交替地放置在磁带 1 和磁带 2 上,直到输入被穷尽为止。然后磁带 1 和磁带 2 被重绕到它们开始的位置,从这些磁带上取出路段再次合并,得到其长度为原来路段长度 2 倍的新路段;这些新路段也像它们形成时那样交替地写到磁带 3 和磁带 4 上(如果磁带 1 包含的路段比磁带 2 多一个,则假定磁带 2 上有一个长度为 0 的额外的虚拟路段),然后所有的磁带都被重绕,而且磁带 3 和磁带 4 的内容被合并成交替地记录到磁带 1 和磁带 2 上的 4 倍长度的路段。这个过程继续进行,每次把路段的长度加倍,直到仅仅剩下一个路段(即完全排好序的文件)为止。如果在内部排序阶段产生了 S 个路段,而且如果 $2^{k-1}<S\leqslant 2^k$,则这个平衡的 2 路合并过程对

所有数据恰好进行了 $k=\lceil \lg S\rceil$ 次合并扫描。

例如,在上述通过容量为 1 000 000 的一个内部存储对 5 000 000 个记录排序的情况下,可知 $S=5$。排序过程的初始分布阶段按如下方式把 5 个路段放置到磁带上:

磁带 1 $R_1\cdots R_{1000000}$;$R_{2000001}\cdots R_{3000000}$;$R_{4000001}\cdots R_{5000000}$

磁带 2 $R_{100001}\cdots R_{2000000}$;$R_{3000001}\cdots R_{4000000}$ (1)

磁带 3 (空)

磁带 4 (空)

然后,第一次合并扫描,按照它读磁带 1 和磁带 2 的顺序,在磁带 3 和磁带 4 上产生更长的路段如下(一个虚拟的路段已经隐式地加在磁带 2 的结尾,使得磁带 1 上的最后的路段 $R_{4000001}\cdots R_{5000000}$ 仅仅拷贝到磁带 3 上):

磁带 3 $R_1\cdots R_{2000000}$;$R_{4000001}\cdots R_{5000000}$ (2)

磁带 4 $R_{2000001}\cdots R_{4000000}$

在所有的磁带被重绕之后,对于数据的下一遍扫描产生(再次简单地拷贝路段 $R_{4000001}\cdots R_{5000000}$;但是如果我们以 8 000 000 个记录开始,则这时磁带 2 就将包含 $R_{4000001}\cdots R_{8000000}$ 了):

磁带 1 $R_1\cdots R_{4000000}$ (3)

磁带 2 $R_{4000001}\cdots R_{5000000}$

最后在另一轮重绕之后,在磁带 3 上产生 $R_1\cdots R_{5000000}$,排序即告完成。

平衡的合并可以容易地推广到对于任何 $T\geqslant 3$ 的 T 条磁带的情形。选择任意数 P,$1\leqslant P<T$,而把 T 条磁带分成两"边",P 条磁带在左边,$T-P$ 条磁带在右边。初始的路段尽可能均匀地分配到左边的 P 条磁带上;然后从左到右进行 P 路合并,紧接着从右到左进行$(T-P)$路合并,等等,直到排序完成为止。P 的最好选择通常认为是$\lceil T/2\rceil$(见习题 3 和 4)。

平衡的两路合并是 $T=4$,$P=2$ 时的特殊情况。用更多条磁带重新考虑上面的例子,取 $T=6$ 和 $T=3$。现在的初始分布是

磁带 1 $R_1\cdots R_{1000000}$;$R_{3000001}\cdots R_{4000000}$

磁带 2 $R_{1000001}\cdots R_{2000001}$;$R_{4000001}\cdots R_{5000000}$ (4)

磁带 3 $R_{2000001}\cdots R_{3000000}$

第一遍合并扫描产生了(在磁带 3 上已经假定有一个虚拟路段)

磁带 4 $R_1\cdots R_{3000000}$

磁带 5 $R_{3000001}\cdots R_{5000000}$ (5)

磁带 6 (空)

第二遍合并扫描完成了这个工作,把 $R_1\cdots R_{5000000}$ 放置到磁带 1 上。在这个特殊的情况下,$T=6$ 实际上和 $T=5$ 是一样的,因为第 6 条磁带仅当 $S\geqslant 7$ 时才使用。

三路合并实际上比两路合并要求更多的计算机处理时间,但是比起读、写和重绕磁带所需要的时间来,一般均可忽略。我们可以仅仅考虑磁带移动的数量,而得

到一个相当好的运行时间的估计。(4)和(5)中的例子只要求对数据扫描两次，而 $T=4$ 时须扫描 3 次。相比之下，$T=6$ 时这个合并所花费的时间仅仅约为前者的 2/3。

平衡合并十分简单，但是如果我们更仔细地观察，则立即发现，它不是处理上边讨论的特殊情形的最好途径！不从(1)进行到(2)并重绕所有的磁带，而是在磁带 3 和磁带 4 分别地包含 $R_1 \cdots R_{2000000}$ 和 $R_{2000001} \cdots R_{4000000}$ 之后，即停止第一遍合并扫描，且使磁带 1 做好读 $R_{4000001} \cdots R_{5000000}$ 的准备。然后重绕磁带 2,3,4，最后通过在磁带 2 上进行三路合并来完成这个排序。则在这个过程中从磁带上读记录的总数量，同在平衡方案中的 5 000 000 + 5 000 000 + 5 000 000 = 15 000 000 相对照，将仅仅是 4 000 000 + 5 000 000 = 9 000 000。一台灵巧的计算机应有能力做到这一点！

其实，当有 5 个路段和 4 条磁带时，通过把它们如下分布，我们甚至能做得更好：

磁带 1　$R_1 \cdots R_{1000000}; R_{3000001} \cdots R_{4000000}$

磁带 2　$R_{1000001} \cdots R_{2000000}; R_{4000001} \cdots R_{5000000}$

磁带 3　$R_{2000001} \cdots R_{3000000}$

磁带 4　(空)

然后对磁带 4 进行三路合并，紧接着进行磁带 3 和磁带 4 的一次重绕，再紧接着进行在磁带 3 上的三路合并，仅仅通过读 3 000 000 + 5 000 000 = 8 000 000 个记录就完成了排序。

而且，当然，如果有 6 条磁带，则可以把初始的路段放置到磁带 1 到磁带 5 上，并且通过对磁带 6 进行五路合并，在一次扫描中就完成排序。这些考虑说明简单的平衡合并不是最好的，故考虑改进的合并型式是有趣的。

本章的以下部分，将更深入地分析外部排序。在 5.4.1 小节中，我们考虑“内部排序”阶段，它产生初始的路段；其中特别有趣的是“替代选择”技术。它利用在大多数数据中存在的次序，来产生其长度实际上大大超过内部存储容量的初始路段。5.4.1 小节也讨论了适合于多路合并的数据结构。

5.4.2 小节到 5.4.5 小节讨论了最重要的合并型式。当我们学习这些型式的特征时，最好先不忙于去跟那些烦人的磁带机和有待排序的实际数据打交道，而是先有一个相当朴素的磁带排序的概念。例如，可以冒失地(如上边所做的那样)假定，原来的输入记录在初始分布阶段神秘地出现；事实上，这些输入记录可能占用 1 条磁带，而且也可能甚至充满若干卷磁带，因为磁带并非无限长！最好是，在对经典的合并样式获得原则性的理解之前，忽略这些过于实际的考虑。在 5.4.6 小节，将讨论强烈地影响合并型式选择的现实中的约束，这个讨论使我们又回到实际中来。5.4.6 小节中，使用在实际中出现的各种各样的假定，来比较 5.4.2 小节到 5.4.5 小节的基本合并型式。

5.4.7 小节和 5.4.8 小节中讨论了不是以合并为基础的解决外部排序的一些其它方法。最后，5.4.9 小节通过讨论诸如磁盘和磁鼓等这样的海量存储的重要排

序问题，完成了对于外部排序的综述。

在最初编写本书时，磁带是大量使用的，而磁盘驱动器还是昂贵的。但在 20 世纪 80 年代，磁盘大为改进，而到了 20 世纪 90 年代末，它们在大多数的计算机系统中，几乎完全替代了磁带机。因此曾经一度是很关键的磁带合并模式的课题，对于当前的需要来说，已变成了只有很有限的关系了。

但是许多模式仍是十分优美的，而且相关的算法反映了早年在计算机科学中所完成的某些最好的研究成果；这些技术实在太漂亮了，不应仓促地扔进历史的垃圾堆去。确实，这些方法中把理论同实际相结合的方式是特别有教益的。因此，以下我们以它们在最后谢幕之前可能是最隆重的出场这样一种方式，对它们进行仔细和完整的讨论。

For all we know now,
these techniques may well become crucial once again.
—PAVEL CURTIS(1997)

习　题

1. [*15*] 正文建议首先进行内部排序，紧接着进行外部合并，为什么不能去掉内部排序阶段，而是从一开始就把记录合并成越来越长的路段？

2. [*10*] 当举例的记录 $R_1R_2\cdots R_{5000000}$ 使用一个 3-磁带平衡方法以 $P=2$ 进行排序时，相当于(1)~(3)磁带的内容序列将是什么？请把这同 4-磁带合并比较；在初始的路段分布好之后，对所有的数据要做多少次扫描？

3. [*20*] 说明当 $P^k(T-P)^{k-1}<S\leqslant P^k(T-P)^k$ 时，应用于 S 个初始路段的平衡的$(P,T-P)$路合并花费 $2k$ 次扫描；而当 $P^k(T-P)^k<S\leqslant P^{k+1}(T-P)^k$ 时，它花费 $2k+1$ 次扫描。

给出对于(a)当 $T=2P$ 时，作为 S 的函数的扫描的精确次数，和(b)对于一般的 P 和 T，当 $S\to\infty$时，扫描的近似次数的一个简单公式。

4. [*HM15*] 对于 $1\leqslant P<T$，P 的什么值使得 $P(T-P)$取极大？

5.4.1 多路合并和替代选择

在 5.2.4 小节中，我们研究了以两路合并为基础的内部排序方法，即把两个有序的序列合并成一个有序序列的过程。不难将其推广成 P 路合并的思想，其中 P 个输入路段被合并成一个输出路段。

假定已经给定 P 个递增的路段，即，其键码按非减次序排列的记录。把它们合并的明显方式，是考察每个路段的第一个记录，选择其中键码最小的那个；然后输出此记录，同时从输入中撤消它；之后重复这一过程。在任何给定的时刻，我们都仅需考虑 P 个键码(每个输入路段一个键码)，并选择最小的。如果有两个或者更多个最小的键码，那就任意选择一个。

当 P 不太大时，通过简单地进行 $P-1$ 次比较来求当前所有键码的最小者，是一种方便的选择方法。但当 P 为 8 或更大时，通过使用 5.2.3 小节所描述的那样一

株选择树，我们就可以节省工作量；于是一旦建立了树，每次就将仅仅需要大约 lg P 次比较。

例如，考虑四路合并的情形，用两级选择树：

```
                                           ┌087┌087  503  ∞
                                           │   └170  908  ∞
步骤 1                                  087┤
                                           │   ┌154  426  653  ∞
                                           └154└612  ∞

                                           ┌170┌503  ∞
                                           │   └170  908  ∞
步骤 2                             087  154┤
                                           │   ┌154  426  653  ∞
                                           └154└612  ∞

                                           ┌170┌503  ∞
                                           │   └170  908  ∞
步骤 3                        087  154  170┤
                                           │   ┌426  653  ∞
                                           └426└612  ∞
  ·
  ·
  ·
                                           ┌∞ ┌∞
                                           │  └∞
步骤 9  087 154 170 426 503 612  653  908 ∞┤
                                           │  ┌∞
                                           └∞ └∞
```

在这个例子中，每个路段的末尾都放置一个附加的键码“∞”，使得合并优雅地结束。由于外部合并一般都涉及非常长的路段，故添加磁带有∞键码的记录后，无论是数据的长度或合并的工作量都不会有实质性的增加，而且这样的“哨兵”记录经常用作一项界定一个文件上路段的有用技术。

在上述过程开始之后，每一步都把最小的元素换成同一路段中随后的元素，并且改变在选择树中对应的路径。于是，该树在步骤 1 中包含 087 的 3 个位置在步骤 2 中被改变；在步骤 2 中包含 154 的 3 个位置在步骤 3 中被改变，等等。在选择树中用另一个键码来代替一个键码的过程，称为替代选择。

我们可以从几个方面来考察这种四路合并。从一种观点看，它等价于像协同子程序那样并发执行的 3 个两路合并；选择树中的每一个节点，都代表一个包含在并发合并过程中的序列。选择树的操作实质上同一个具有“最小者先出”原则的优先队列一样。

如同 5.2.3 小节一样，我们可以使用一个堆代替一株选择树，以实现优先队列（当然堆将安排成最小元素出现在顶上，而不是最大的元素在顶上，应颠倒等式 5.2.3-(3)的顺序）。由于一个堆并没有固定的大小，因此我们避免使用“∞”键码；当堆成空时，合并完成。另一方面，应用外部排序时通常都要处理相当长的记录和

键码,故堆中填入指向键码的指针以代替键码本身;下边我们将看到,选择树可以以这样一种方便的方式由指针表示,在这种情况下,它们可能优于堆。

"失利者"的树 图 62 所示为具有 12 个外部(方框)节点和 11 个内部(圆圈)节点的完备二叉树;如果把这株树看做是选择最小键码的一次锦标赛的话,则外部节点中已填入键码,而内部节点中已经填入"胜利者"。每个节点上边小号数字标明了为完备的二叉树分配连续存储位置的传统方法。

当最小的键码 061 用图 62 中选择树的另一个键码来代替时,为确定选择树的新状态我们需要考察键码 512,087 和 154,而非其它现有的键码。把此树看做一场锦标赛,这 3 个键码就是在竞赛中同 061 对垒的失利者。这提示我们,在这株树的内部节点中真正应该存储的是每次对垒的失利者,而不是胜利者;这样,为更新这株树所需要的信息就很容易得到了。

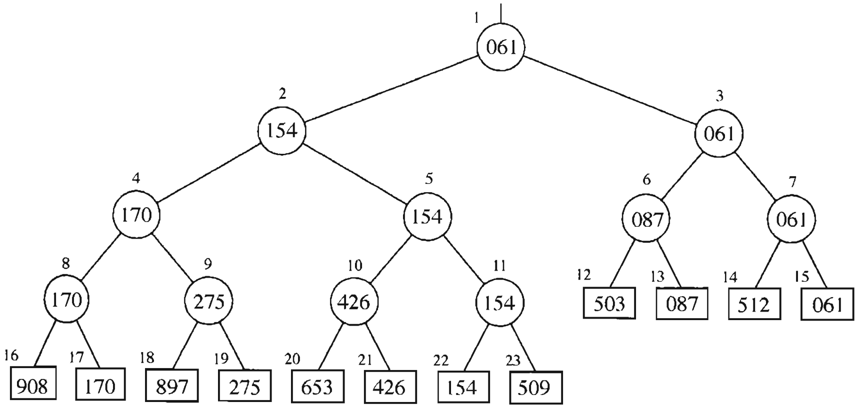


图 62 使用一个其节点编号为 1 到 23 的二叉树来选择最小键码的一次锦标赛

图 63 为与图 62 同样的树,但它表示失利者而不是胜利者。在这株树的顶部已经附加了一个额外的数 0,指出这场锦标赛的冠军。注意,除了冠军之外每个键码都恰巧是一次比赛的失败者(参见 5.3.3 小节),所以每个键码都在一个外部节点中出现一次和在一个内部节点中出现一次。

实际上,图 63 底部的外部节点表示存于计算机存储器中的相当长的记录,而内部节点表示指向这些记录的指针。注意,P 路合并恰巧调用 P 个外部节点和 P 个内部节点,每个都在相邻的组中,于是这本身就提示了若干有效的存储分配方法。不难看出,如何利用替代选择的"面向失利者"的树;我们将在本节的稍后部分稍微详细地讨论这个算法。

由替代选择所产生的初始路段 如果基本上用输入数据本身进行 P 路合并,则替代选择技术也可以用于外部排序的第一阶段!在这种情况下,把 P 取得相当大,使得内部存储器基本上已被充满。当输出一个记录时,它就被下一个输入记录

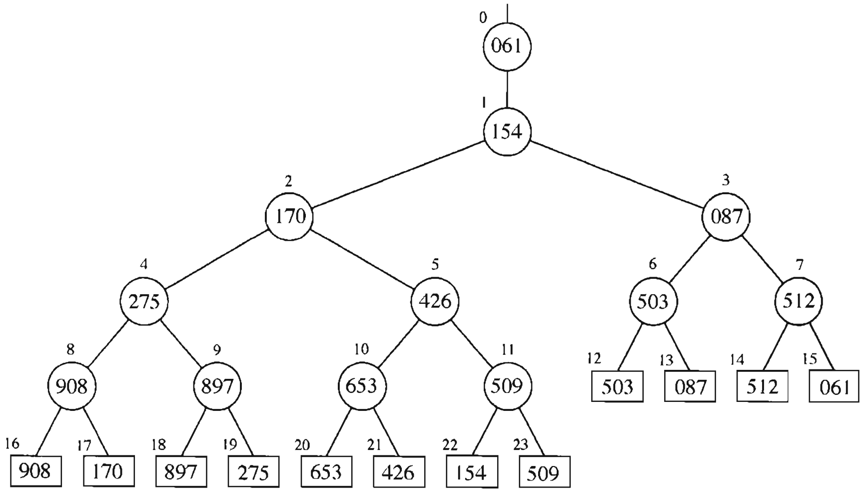


图 63 和图 62 相同的锦标赛,但显示的是失利者而不是胜利者;
冠军出现于最顶上

所代替。如果新记录的键码小于刚才输出的键码,则不能把它包括在当前的路段中;否则可以用通常的方式把它送入选择树中,并且它将形成当前正在产生的路段的一部分。于是每个路段就能包含多于 P 个记录,尽管任何时候在选择树中决没有多于 P 个记录。表 1 所示为 $P=4$ 的这一过程:磁带圆括号的数将要包括进随后的路段中。

表 1 四路替代选择的例子

存储内容				输出
503	087	512	061	061
503	087	512	908	087
503	170	512	908	170
503	897	512	908	503
(275)	897	512	908	512
(275)	897	653	908	653
(275)	897	(426)	908	897
(275)	(154)	(426)	908	908
(275)	(154)	(426)	(509)	(路段结束)
275	154	426	509	154
275	612	426	509	275
		等等		

形成初始路段的这一重要方法,首先是由 Harold H. Seward 描述的[硕士论文,Digital Computer Laboratory Report R-232(Mass. Inst. of Technology, 1954), 29～30],他给出的理由使人们确信,当应用于随机数据时,这些路段将包含 $1.5P$ 个以

上的记录。1950 年左右,A.I.Dumey 在谈到工程研究协会设计的一部特殊排序设备时,也提出了这个思想,但他没有发表。“替代选择”这一名称,是由 E.H.Friend 杜撰的[*JACM* **3**(1956),154],他解释说:“提不出所产生序列的预期长度的公式,但是实验提示 $2P$ 是一个合理的预测。”

为了说明 $2P$ 确是预期的路段长度,E.F.Moore 发现了一种聪明的方式,他把这个情况同在一个圆形轨道上的扫雪机做了比较(*U.S. Patent 2983904* (1961),columns 3～4)。考虑图 64 中所示的情况;雪片均匀地落在一条圆形的路上,一台孤独的扫雪机不断地清扫雪。一旦已经把雪扫出路外,它就从这个系统消失。可以通过实数 x 来指明路标,$0\leqslant x<1$;落在位置 x 的雪片表示其键码为 x 的输入记录,而扫雪机表示替代选择的输出。扫雪机的基本速度同它所遇到的雪的高度成反比,且情况是完全平衡的,即在路上雪的总量在所有时刻都恰巧是 P。每当扫雪机通过点 0 时,便在输出中形成一个新的路段。

在这个系统已经运行一段时间之后,显然,直观上它将趋于一个稳定状态,在这个状态下,扫雪机以恒速运行(由于这个轨道的圆形对称性)。这意味着当遇到扫雪机时,雪处于常数高度中,而且如图 65 所示,这高度在扫雪机前面线性地降低。由此推出,在一个循环(即路段长度)中所扫除的雪的体积是任何一个时候存在的量(即 P)的两倍。

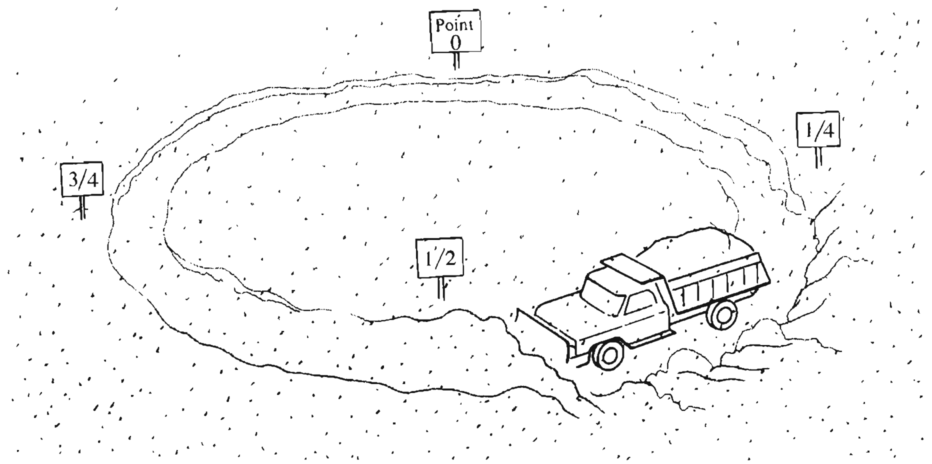


图 64 在环形圆圈上连续不停运行的扫雪车

在许多商业应用中输入数据不是完全随机的;它已经有某种程度的既定次序。因此由替代选择产生的路段很可能包含甚至 $2P$ 个以上的记录。我们将看到,外部合并排序所需要的时间在很大程度上受初始分布阶段所产生的路段数的支配,因而替代选择显得特别可取;其它类型的内部排序由于存储大小的限制,将产生大约两倍之多的初始路段。

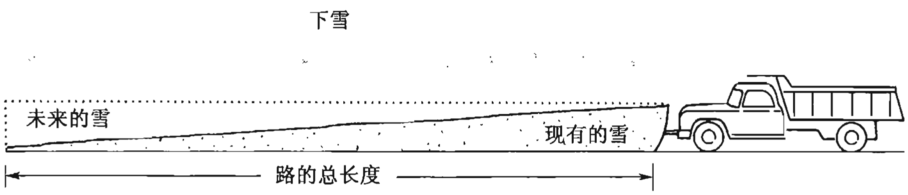


图 65 横断面,示出当这个系统处于它的"稳定状态"时机犁前雪的高度变化

现在让我们详细考虑由替代选择建立初始路段的过程。下列算法是由 John R. Walters,James Painter 及 Martin Zalk 给出的,他们在 1958 年把它用于 Philco2000 的合并路段程序中。它加入了一个很好的方法,通过相当简单和一致的逻辑来建立选择树的初态,并区分了属于不同路段的记录,以及清理最后一个路段(由替代选择产生的最后路段的适当处理,是带点窍门的,对于程序员说来它势必是一个令人困惑的难点)。基本的思想是把每个键码都当作一个对偶(S,K),其中 K 是原来的键码,而 S 是这个记录所属路段的编号,当这样扩展的键码按字典次序排列,并以 S 作主键码,以 K 作辅键码时,我们就得到由替代选择产生的输出序列。

以下的算法使用了包含 P 个节点的一个数据结构来表示选择树;假定第 $j(0\leqslant j<P)$个节点 $X[j]$包含从 LOC$(X[j])=L_0+cj$ 开始的 c 个字,它既表示图 63 中的内部节点号 j,又表示外部节点号 $P+j$。在每个节点中有若干命名了的字段:

KEY = 存储在这个外部节点中的键码;

RECORD = 存储在这个外部节点中的记录(包括 KEY 作为一个子字段);

LOSER = 指向存储在这个内部节点中的"失利者"的指针;

RN = 由 LOSER 指出的记录路段号;

PE = 指向这株树中在这个外部节点上方的内部节点的指针;

PI = 指向这株树中在这个内部节点上方的内部节点的指针。

例如,当 $P=12$ 时,图 63 的内部节点号 5 和外部节点号 17 都将通过字段 KEY $=170$,LOSER $=L_0+9c$(外部节点号 21 的地址),PE $=L_0+8c$,PI $=L_0+2c$ 在 $X[5]$ 中表示出来。

字段 PE 和 PI 有常数值,所以它们不必明显地出现于存储器中;然而,外部排序的初始阶段,有时跟不上输入/输出设备的速度,故通常值得把这些多余的值作为数据存起来,而不是每次重新计算它们。

算法 R(替代选择) 这个算法顺序地从一个输入文件读入记录,并把它们顺序地写在一个输出文件上,产生 RMAX 个路段,除最后的路段外,每个路段的长度均大于或等于 P。共有 $P\geqslant 2$ 个节点 $X[0],\cdots,X[P-1]$,每个节点所含的字段如上所述。

R1.[初始化] 置 RMAX←0,RC←0,LASTKEY←∞,Q←LOC$(X[0])$,以及 RQ←0 (RC 为当前路段的编号,而 LASTKEY 为最后输出记录的键码。LASTKEY 的初

值应当大于任何可能的键码;参见习题8)。对于任何 $0 \leqslant j < P$,当 $J = \text{LOC}(X[j])$时,置 $X[j]$的初始内容如下:

$$\text{LOSER(J)} \leftarrow \text{J};\ \text{RN(J)} \leftarrow 0;$$

$$\text{PE(J)} \leftarrow \text{LOC}(X[\lfloor (P+j)/2 \rfloor]);\ \text{PI(J)} \leftarrow \text{LOC}(X[\lfloor j/2 \rfloor])$$

(LOSER(J)和 RN(J)的赋值都是人为的办法,为的是通过考虑一个虚构的编号为0的路段来建立树的初态。该路段决不会输出,这只是一个技巧,见习题10。)

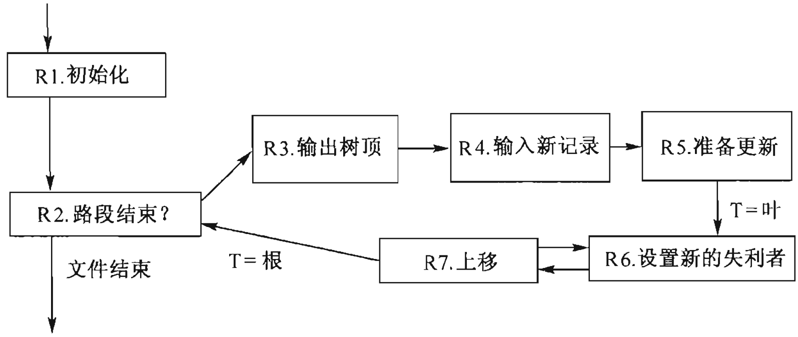


图66 用替代选择作初始路段

R2.[路段结束?] 如果 RQ=RC,则继续转到步骤R3。(否则 RQ=RC+1,此时我们刚刚完成编号为RC的路段;一个合并型式所要求的对于排序的随后扫描的任何特殊动作将在这时完成)。如果 RQ>RMAX,则停止,否则置 RC←RQ。

R3.[输出树顶](现在Q指向"冠军",而RQ是它的路段号)如果 RQ≠0,则输出 RECORD(Q),并置 LASTKEY←KEY(Q)。

R4.[输入新记录] 如果输入文件穷尽了,则置 RQ←RMAX+1 且继续转到步骤R5。否则置 RECORD(Q)为输入文件的下个记录。如果 KEY(Q)<LASTKEY(于是这个新记录不属于当前的路段),则置 RQ←RQ+1,然后如果 RQ>RMAX,则置 RMAX←RQ。

R5.[准备更新](现在Q指向一个新记录,其路段号是RQ) 置 T←PE(Q)(T是一个指针变量,它将沿树上移)。

R6.[置新的失利者] 如果 RN(T)<RQ 或者如果 RN(T)=RQ,并且 KEY(LOSER(T))<KEY(Q),则对换 LOSER(T)↔Q,RN(T)↔RQ(变量Q和RQ记住当前的胜利者和它的路段号)。

R7.[上移] 如果 T=LOC(X[1])则返回到R2,否则置 T←PI(T)并返回R6。
▌

算法R中的输入和输出一次只涉及一个记录,然而实际上最好是读和写相当大块的记录。因此,存储器中有某些输入和输出缓冲区,它们实际上在幕后降低 P 的大小,我们在5.4.6小节说明这一点。

* **路段的延迟重新组成** 使用我们称之为自由度的概念，R.J.Dinsmore[*CACM* **8**(1965),48] 提出了改进替代选择的一个非常有趣的方法。如同我们已经看到的，磁带上在一个路段之内的每组记录是按非减次序排好的，于是它的头一个元素是最低的而最后元素是最高的。在通常的替代选择过程中，一个路段内每组的最低元素决不小于该路段中前面那组的最高元素；这是“1 个自由度”。Dinsmore 提议把这条件放松成为“m 个自由度”，其中每组的最小元素可以小于前一组的最高元素，只要*它不小于本路段前的 m 个不同的组中的最高元素就行*。如同以前一样，在单个组内的记录是有序的，但相邻的组不必是有序的。

例如，假设每组恰有两个记录；下列组序列是具有 3 个自由度的一个路段：

$$|\ 08\quad 50\ |\ 06\quad 90\ |\ 17\quad 27\ |\ 42\quad 67\ |\ 51\quad 89\ | \tag{1}$$

该路段将要包含的下一个组，必须以一个不小于{50,90,27,67,89}的第 3 个最大元素(即 67)的元素开始。如果仅有两个自由度的话，序列(1)将不是一个路段，因为 17 既小于 50，也小于 90。

具有 m 个自由度的一个路段，当它在排序的下一个阶段被读入时，可以被“重新组成”，使得对于所有实用的目的说来，它都是通常意义下的一个路段。我们从把 m 个组读到 m 个缓冲区开始，对它们进行 m 路合并；当穷尽了一个缓冲区时，以第 $m+1$ 个组代替它，等等。用这种办法，我们可以重新把这一路段恢复成一单个序列，因为每个新近读的头一个字都必须大于或等于刚穷尽的组的最后一个字(免得它小于在它之前的 m 个不同组中的最高元素)。这个重新组成路段的方法，实际上就像对于所有的输入组都使用一台磁带机的 m 路合并一样！重新组成过程的工作方式像一个协同子程序，它被调用来一次发送路段的一个记录。我们可以对从不同的磁带机上来的具有不同自由度的路段加以重新组成，并且合并得到的路段，所有这些都在同一个时间进行，这种方法实际上就像本节开始所说明的，四路合并可以想像为一次进行若干个两路合并一样。

对这个有独创性的思想难以进行精确分析，但 T.O.Espelid 已经说明了如何来扩充扫雪机的类似性，以得到对于这个特性的一个近似公式[*BIT* **16** (1976),133～142]。根据这个同经验测试很一致的公式，当 b 是块大小且 $m\geqslant 2$ 时，路段长度将大约是

$$2P+(m-1.5)\left(\frac{2P+(m-2)b}{2P+(2m-3)b}\right)b$$

这样一种增加尚不足以证明由此而增加的复杂性是合理的；另一方面，在第二个排序阶段，如果有足够的地方可安排相当大量的缓冲区，它可能是有利的。

* **自然的选择** W.D.Frazer 和 C.K.Wong(黄泽权)[*CACM* **15** (1972),910～913]剖析了增加由替代选择产生的路段长度的另一方式。他们的想法是像算法 R 那样做，但当一个新记录的键码小于 LASTKEY 时，这个新的记录被输出到一个外部库中，并读入另一个新的记录。这个过程延续到这个库被一定数量的记录 P' 充满为止；这时，当前路段的剩下部分从该树中输出，而存于库中的项目即用作下一路段

的输入。

库的使用趋向于产生比替代选择更长的路段，因为它避免了属于下个串的“死的”记录，不让它们来充斥该树；但它需要额外的时间以从库进行输入和输出到库中。当 $P'>P$ 时，某些记录有可能两次入库，但当 $P'\leqslant P$ 时，这将绝不可能发生。

Frazer 和 Wong 对他们的方法进行了广泛的实验测试，得知当 P 相当大（比如说 $P\geqslant 32$）和 $P'=P$ 时，随机数据的平均路段长度由 eP 给出，其中 $e=2.718$ 是自然对数底。由于这个现象，以及由于这个方法是对于简单的替代选择的一种渐进改良，自然地使他们把他们的方法称为*自然的选择*。

通过再次考虑图 64 的扫雪机并且应用初等微积分，可以证明路段长度的“自然”定律。设 L 是道路的长度，并设 $x(t)$ 是在时间 $t(0\leqslant t\leqslant T)$ 时扫雪机的位置。当雪暂时停止而扫雪机回到它开始的位置（清除了在它的通路上剩下的 P 个单位的雪）时，假定在时间 T 时库是满的。除了“平衡条件”不同外，这个情况和以前是一样的；代替在所有时刻道路上有 P 个单位的雪，我们有 P 个单位的雪在扫雪机之前，而且库（在扫雪机之后）增加成 $P'=P$ 个单位。如果 $h(x,t)dx$ 个记录被输出，$h(x,t)$ 是在时间 t 时的雪的高度，且位置 $x=x(t)$（这里采用适当的单位），则在一个时间间隔 dt 内扫雪机前进 dx。

因此，对所有 x，$h(x,t)=h(x,0)+Kt$，其中 K 是落雪的速度。由于在存储器中的记录数保持不变，所以 $h(x,t)$ 也就是在扫雪机之前输入的记录数，即 $Kdt(L-x)$（见图 67）。于是

$$\frac{dx}{dt}=\frac{K(L-x)}{h(x,t)} \tag{2}$$

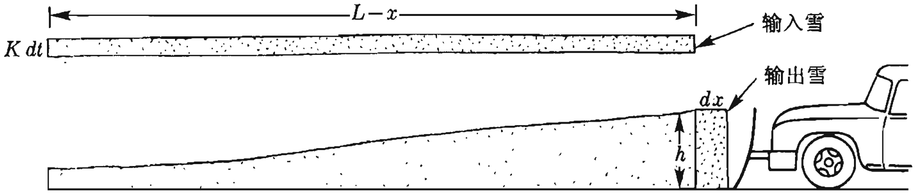


图 67　等量的雪被输入和输出；扫雪机在时间 dt 内移动 dx

幸而，结果证明每当 $x=x(t)$ 和 $0\leqslant t\leqslant T$ 时，$h(x,t)$ 是等于 KT 的常数，因为在扫雪机通过点 $x(t)$ 后，雪稳定地降落到该位置达 $T-t$ 个时间单位，在它返回前还要加上 t 个时间单位。换句话说，在已达到稳定状态的前提下，每个旅程都是一样的，扫雪机看到在它的旅程中所有的雪都处于同样的高度，因此清扫的雪的总量（路段长度）是 LKT；而存储器中雪的数量是在时间 T 后清除的数量，即 $KT(L-x(T))$。使得 $x(0)=0$ 的(2)的解为

$$x(t)=L(1-e^{-t/T}) \tag{3}$$

因此 $P=LKTe^{-1}=$（路段长度）$/e$；而这就是我们所要证明的。

习题 21～23 说明，这个分析可以推广到一般的 P' 的情况；例如，当 $P'=2P$ 时，

平均路段长度得知是 $e^{\theta}(e-\theta)P$，其中 $\theta=\left(e-\sqrt{e^2-4}\right)/2$，这大概是还没有人能立即猜出的结果；表 2 说明了路段长度对于库大小的依赖性；通过查这张表，在一个给定的计算机环境中自然选择的有用性即可估计出来。对于 $<P$ 的库大小的表项，使用习题 27 中一个改进的技术。

如同 T.C.Ting（丁子锦）和 Y.W.Wang（王亚威）在*Comp.J.* **20** (1977)，298～301 中所讨论的那样，延迟路段重新组成和自然选择的思想可以联合在一起。

表 2 自然选择产生的路段长度

库大小	路段长度	$k+\theta$	库大小	路段长度	$k+\theta$
0.10000P	2.15780P	0.32071	0.00000P	2.00000P	0.00000
0.50000P	2.54658P	0.69952	0.43428P	2.50000P	0.65348
1.00000P	2.71828P	1.00000	1.30432P	3.00000P	1.15881
2.00000P	3.53487P	1.43867	1.95014P	3.50000P	1.42106
3.00000P	4.16220P	1.74773	2.72294P	4.00000P	1.66862
4.00000P	4.69446P	2.01212	4.63853P	5.00000P	2.16714
5.00000P	5.16369P	2.24938	21.72222P	10.00000P	4.66667
10.00000P	7.00877P	3.17122	5.29143P	5.29143P	2.31329

注：在习题 22 或习题 27（当 $k=0$）中定义了量 $k+\theta$

***替代选择的分析** 现在让我们转回到没有辅助库的替代选择的情况。扫雪机的类比，给了我们在稳定状态下由替代选择所得到的平均路段长度的一个相当好的估计，但是通过应用我们在 5.1.3 小节中做的关于排列中路段的研究的事实，有可能获得算法 R 的许多更精确的信息。为此目的，假定输入文件是 0 和 1 之间的独立随机实数的一个任意长的序列，是更为方便的。

设

$$g_P(z_1,z_2,\cdots,z_k)=\sum_{l_1,l_2,\cdots,l_k\geq 0}a_P(l_1,l_2,\cdots,l_k)z_1^{l_1}z_2^{l_2}\cdots z_k^{l_k}$$

是由在这样一个文件上的 P 路替代选择所产生的路段长度的生成函数，其中 $a_P(l_1,l_2,\cdots,l_k)$ 是第一个路段长度为 l_1，第二个长度为 l_2，…，第 k 个长度为 l_k 的概率。下列“独立性定理”是一个基本的定理，因为它把这个分析归结为 $P=1$ 的情况：

定理 K $g_P(z_1,z_2,\cdots,z_k)=g_1(z_1,z_2,\cdots,z_k)^P$

证明 设输入键码为 $X_1,X_2,X_3,\cdots$。按照它们在这株树中所处的外部节点的位置，算法 R 把它们划分为 P 个子序列；包含 X_n 的子序列由值 $X_1,\cdots,X_n$ 确定。因此每一个子序列是 0 和 1 间独立的随机数的一个独立序列。进而，替代选择的输出恰是对这些子序列进行一次 P 路合并所应得到的；一个元素属于一个子序列的

第 j 个路段的充要条件是,它属于由替代选择产生的第 j 个路段(因为在步骤 R4 中,LASTKEY 和 KEY(Q)属于同一个子序列)。

换句话说,我们也完全可以假定,算法 R 正被应用于 P 个独立的随机输入文件,而且步骤 R4 从对应于外部节点 Q 的文件读下一个记录;在这个意义下,该算法等价于一个 P 路合并,并以"下坡"来标志诸路段的结束。

于是,当且仅当诸子序列有长度分别为$(l_{11},\cdots,l_{1k}),\cdots,(l_{P1},\cdots,l_{Pk})$的路段时,输出中有长度为$(l_1,\cdots,l_k)$的路段。其中 l_{ij}是满足 $\sum_{1\leqslant i\leqslant P} l_{ij}=l_j$ 的某些非负整数,$1\leqslant j\leqslant k$。由此得出

$$a_P(l_1,\cdots,l_k)=\sum_{\substack{l_{11}+\cdots+l_{P1}=l_1\\ \vdots\\ l_{1k}+\cdots+l_{Pk}=l_k}} a_1(l_{11},\cdots,l_{1k})\cdots a_1(l_{P1},\cdots,l_{Pk})$$

而这等价于所求的结果。 ▌

我们已经在 5.1.3 小节讨论了当 $P=1$ 时,第 k 个路段的平均长度 L_k,在表 5.1.3-2 中列出了这些值。定理 K 蕴含对于一般的 P,第 k 个路段的平均长度是当 $P=1$ 时的平均长度的 P 倍,即 L_kP;而且方差也是 P 倍,所以路段长度的标准差同 $\sqrt{P}$成比例。这些结果是 1958 年左右由 B.J.Gassner 首先导出的。

于是,对于随机数据,由算法 R 产生的头一个路段大约包含$(e-1)P\approx1.718P$个记录。第二个路段,大约是$(e^2-2e)P\approx1.952P$ 个记录。第三个,大约是 $1.996P$;而随后的路段,将非常接近于包含 $2P$ 个记录,直到我们得到最后两个路段(见习题 14)为止。这些路段长度的大多数标准差都近似于$\sqrt{(4e-10)P}\approx0.934\sqrt{P}$[*CACM* **6** (1963),685~687]。进而,习题 5.1.3-10 说明,前 k 个路段的总长度将相当接近于$\left(2k-\frac{1}{3}\right)P$,并有$\left(\left(\frac{2}{3}k+\frac{2}{9}\right)P\right)^{1/2}$的标准差。在习题 5.1.3-9 和 11 中导出了生成函数 $g_1(z,z,\cdots,z)$和 $g_1(1,\cdots,1,z)$。

上边的分析已经假定输入文件是无限长的,但定理 K 的证明表明,在至少包含 $l_1+\cdots+l_k+P$ 个元素的任何随机输入序列中,都将得到相同的概率 $a_p(l_1,\cdots,l_k)$。所以上述结果在小的标准差的观点下,对于比如说大小为 $N>(2k+1)P$ 的文件是可应用的。

我们将看到某些应用,其中合并型式要求某些路段是递增的,而某些是递减的,由于在一个递增的路段结束时,在存储器中累积的剩余往往包含平均比随机数更小些的数,故顺序方向的改变减少了路段的平均长度。例如,考虑一台扫雪机,在它每次到达一条直路的终点时必须跑一个 U 形转弯;它将非常快速地通过刚才扫过的区域。当颠倒方向时,随机数据的路段长度将在 $1.5P$ 和 $2P$ 之间变动(见习题 24)。

习 题

1. [*10*] 在本节开头的四路合并例子中,步骤 4 是什么?

2. [*12*] 如果以 612 代替键码 061,对于图 63 的树将做什么变动?

3. [*16*] (E.F.Moore) 当应用四路替代选择于下列一串单词时,产生的输出是什么?

fourscore and seven years ago our fathers brought forth on this continent a new nation conceived in liberty and dedicatd to the proposition that all men are created equal.

使用通常的字母顺序,把每个词处理作一个键码。

4. [*16*] 把四路自然选择应用于习题 3 的句子,使用容量为 4 的一个库。

5. [*00*] 真或假:仅当 P 是 2 的乘方时或者是两个 2 的乘方之和时,使用一个树的替代选择才有效。

6. [*15*] 算法 R 指明,必须 $P\geqslant 2$;应如何稍微修改这个算法,以使它对于所有 $P\geqslant 1$ 成立?

7. [*17*] 当没有任何输入时,算法 R 做什么?

8. [*20*] 算法 R 利用了一个人为的键码"∞",它必须大于任何可能的键码。证明:要是真有一个键码等于∞的话,这个算法可能失误,并说明在真正的∞的实现并不方便的情况下,如何来修改这个算法?

▶9. [*23*] 你将怎样修改算法 R,使得某些指定的路段(依赖于 RC)按递增次序输出,而其它的按递减次序输出?

10. [*26*] 在步骤 R1 中,LOSER 指针的初态通常不对应于任何实际的锦标赛,因为外部节点 $P+j$ 不会处于内部节点 j 之下的子树中,说明为什么算法 R 仍然有效[提示:如果在步骤 R1 中,{LOSER(LOC($X[0]$)),…,LOSER(LOC($X[P-1]$))}被置成{LOC($X[0]$),…,LOC($X[P-1]$)}的任何一个排列,则这个算法是否有效]。

11. [*M25*] 真或假:假定输入是随机的,在步骤 R4 中 KEY(Q)<LASTKEY 的概率近似于$\frac{1}{2}$。

12. [*M46*] 详细分析算法 R 的每一部分被执行的次数,例如,在步骤 R6 中所做的交换有多频繁?

13. [*13*] 为什么由替代选择产生的第二个路段通常都大于第一个路段?

▶14. [*HM25*] 用扫雪机的类比来估计,由输入数据的一个长序列通过替代选择所产生的最后两个路段的平均长度。

15. [*20*] 真或假:由替代选择产生的最后路段决不包含多于 P 个记录,请讨论你的答案。

16. [*M26*] 试求一个文件 $R_1R_2\cdots R_N$ 将由 P 路替代选择在一次扫描中完全排序的一个"简单的"必要和充分条件。当输入是{1,2,…,N}的随机排列时,作为 P 和 N 的函数发生这种情况的概率是什么?

17. [*20*] 当输入键码处于递减次序下,即 $K_1\geqslant K_2\geqslant\cdots\geqslant K_N$ 时,由算法 R 产生的输出是什么?

▶18. [*22*] 如果算法 R 再次应用于由算法 R 产生的一个输出文件,则将发生什么情况?

19. [*HM22*] 用扫雪机的类比来证明,由替代选择产生的第一个路段的长度近似为$(e-1)P$ 个记录。

20. [*HM24*] 当 $P=P'$ 时,由自然选择产生的第一个路段约为多长?

▶21. [*HM23*] 试确定当 $P'<P$ 时,由自然选择产生的路段的近似长度。

22. [*HM40*] 这一习题的目的是确定当 $P'>P$ 时，在自然选择中得到的平均路段长度。令 $\kappa=k+\theta$ 是一个 $\geqslant 1$ 的实数，其中 $k=\lfloor\kappa\rfloor$ 和 $\theta=\kappa \bmod 1$，并考虑函数 $F(\kappa)=F_k(\theta)$，其中 $F_k(\theta)$ 是由生成函数

$$\sum_{k\geqslant 0}F_k(\theta)z^k = e^{-\theta z}/(1-ze^{1-z})$$

定义的多项式，于是 $F_0(\theta)=1, F_1(\theta)=e-\theta, F_2(\theta)=e^2-e-e\theta+\frac{1}{2}\theta^2$，等等。

假设一台扫雪机在时间 0 启动，开始模拟自然选择的过程，并假设在 T 个时间单位之后，恰有 P 个雪花已经落在它后面。这时，第二台扫雪机开始相同的旅程，它在时间 $t+T$ 时所占的位置就是第一台扫雪机在时间 t 时所占的位置。最后，在时间 κT，恰有 P' 个雪花落在第一台扫雪机之后；它在一瞬间把剩下一段路全扫完，随之即告消失。

利用这一模型来表示自然选择的过程，说明当

$$P'/P = k+1+e^{\theta}\Big(\kappa F(\kappa)-\sum_{j=0}^{k}F(\kappa-j)\Big)$$

时，得到一个长度等于 $e^{\theta}F(\kappa)P$ 的路段。

23. [*HM35*] 上题分析了当记录按同一次序(即先进先出)写入和读出库时的自然选择。如果以完全随机的次序读入上一路段存进库中的内容，就像库中的记录在两个路段之间已整个地重"洗"了一样，试求应得到的近似路段长度。

24. [*HM39*] 本题的目的是分析因偶然改变替代选择中路段的方向所引起的效果。

a)设 $g_P(z_1,z_2,\cdots,z_k)$ 是如同在定理 K 中那样定义的一个生成函数，但同时，k 个路段中的每个都已被确定是递增还是递减的。例如，我们可以说所有奇数编号的路段是递增的，所有偶数编号的路段是递减的。试说明对于这种类型的 2^k 个生成函数的每一个，定理 K 都成立。

b)作为 a)的推论，我们可以假设 $P=1$。我们也可以假定，输入是 0 和 1 之间的独立随机数的一个一致分布序列。设

$$a(x,y)=\begin{cases}e^{1-x}-e^{y-x} & \text{如果 } x\leqslant y\\ e^{1-x} & \text{如果 } x>y\end{cases}$$

若给定某个递增路段以 x 开始的概率为 $f(x)dx$，试证明 $\left(\int_0^1 a(x,y)f(x)dx\right)dy$ 为下一个路段以 y 开始的概率[提示：当 x 和 y 已知时，对于每个 $n\geqslant 0$，考虑 $x\leqslant X_1\leqslant\cdots\leqslant X_n>y$ 的概率]。

c)考虑以概率 p 改变方向的路段；换句话说，在第一个路段后每个路段的方向，有 $q=(1-p)$ 的机会被随机地选择成和以前的路段一样，而选择相反方向的机会是 p(于是当 $p=0$ 时，所有的路段都有相同的方向；当 $p=1$ 时，这些路段都改变方向；而当 $p=\frac{1}{2}$ 时，这些路段是独立地随机的)。设

$$f_1(x)=1, f_{n+1}(y)=p\int_0^1 a(x,y)f_n(1-x)dx+q\int_0^1 a(x,y)f_n(x)dx$$

说明当第 $(n-1)$ 个路段递增时第 n 个路段以 x 开始的概率为 $f_n(x)dx$，当第 $(n-1)$ 个路段递减时第 n 个路段以 x 开始的概率为 $f_n(1-x)dx$。

d)求对于"稳态"方程

$$f(y)=p\int_0^1 a(x,y)f(1-x)dx+q\int_0^1 a(x,y)f(x)dx, \int_0^1 f(x)dx=1$$

的一个解[提示：证明 $f''(x)$ 是独立于 x 的]。

e)证明 c)中的序列 $f_n(x)$ 相当迅速地收敛到 d)中的函数。

f)证明,以 x 开始的一个递增路段的平均长度为 e^{1-x}。

g)最后,把上述这些结果归结到一起,以证明下列定理:在替代选择中,*如果连续一系列路段的方向是独立地以概率 p 逆转的,则平均路段长度趋于* $(6/(3+p))P$(关于此定理的 $p=1$ 的情况,首先是由 Knuth 导出的[*CACM* **6**(1963),685～688];$p=\frac{1}{2}$ 的情况则首先由 A.G.Konheim 于 1970 年加以证明)。

25. [*HM40*] 考虑下列过程:

N1. 读一个记录到容量仅为一个字的“库”中,然后读另一个记录 R 并令 K 是它的键码。

N2. 输出这个库,置 LASTKEY 为它的键码,并置这个库为空。

N3. 如果 K<LASTKEY,则输出 R 并置 LASTKEY←K,然后转到 N5。

N4. 如果库非空,则转到 N2;否则把 R 送入库中。

N5. 读入新记录 R,并设 K 是它的键码。转到 N3。 ▌

这实际上等价于 $P=1$ 和 $P'=1$ 或 2 的自然选择(取决于你是选定在库满的时刻来弄空它,还是在它大约溢出了的时刻弄空它),惟一的区别是它产生递减的路段,而且它决不停止。对于本题的目的说来,后边的异常情况是方便的和无害的假定。

如同在习题 24 中那样进行,设 $f_n(x,y)\mathrm{d}y\mathrm{d}x$ 是恰在第 n 次执行步骤 N2 之后(LASTKEY 和 K 的值分别是 x 和 y)的概率。证明存在一个变量的函数 $g_n(x)$,使得当 $x<y$ 时 $f_n(x,y)=g_n(x)$,而当 $x>y$ 时 $f_n(x,y)=g_n(x)-e^{-y}(g_n(x)-g_n(y))$。这个函数 $g_n(x)$ 由关系式 $g_1(x)=1$,

$$g_{n+1}(x)=\int_0^x e^u g_n(u)\mathrm{d}u+\int_0^x \mathrm{d}v(v+1)\int_v^1 \mathrm{d}u((e^v-1)g_n(u)+g_n(v))+$$
$$x\int_x^1 \mathrm{d}v\int_v^1 \mathrm{d}u((e^v-1)g_n(u)+g_n(v))$$

来定义。进一步说明,第 n 个路段的预期长度为

$$\int_0^1 \mathrm{d}x\int_0^x \mathrm{d}y(g_n(x)(e^y-1)+g_n(y))\left(2-\frac{1}{2}y^2\right)+\int_0^1 \mathrm{d}x(1-x)g_n(x)e^x$$

[注意:这些方程的“稳定状态”的解显得非常复杂;它的数值解已经为 J.McKenna 得到,他说明这些路段长度趋于 2.61307209 的极限值。定理 K 不能应用于自然选择,所以 $P=1$ 的情况对其它的 P 行不通]。

26. [*M33*] 把习题 25 中的算法当作 $P'=1$ 时自然选择的定义,对任何 $r\geqslant 0$,求当 $P'=r$ 时第一个路段的预期长度如下

a)证明第一个路段长度为 n 的概率是

$$(n+r)\left[{n+r \atop n}\right]\Big/(n+r+1)!$$

b)通过规则

$$\left[\left[{0 \atop m}\right]\right]=\delta_{m0},\quad \left[\left[{n \atop m}\right]\right]=(n+m-1)\left(\left[\left[{n-1 \atop m}\right]\right]+\left[\left[{n-1 \atop m-1}\right]\right]\right) \text{ 对 } n>0$$

定义“相关联的斯特林数”$\left[\left[{n \atop m}\right]\right]$,证明

$$\left[{n+r \atop n}\right]=\sum_{k=0}^{r}\binom{n+r}{k+r}\left[\left[{r \atop k}\right]\right]$$

c)证明第一个路段的平均长度因此是 $c_r e - r - 1$,其中

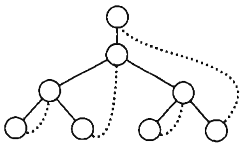
$$c_r = \sum_{k=0}^{r} \left[\left[\begin{matrix} r \\ k \end{matrix} \right]\right] \frac{r+k+1}{(r+k)!}$$

▶27. [*HM30*] (W. Dobosiewicz) 当对于 $P' < P$ 使用自然选择时,在库变满时我们不必停下形成一个路段;像在替代选择中一样,我们可以把不属于当前路段的记录存进主优先队列中,直到剩下当前路段的 P' 个记录为止。于是我们可以把它们"喷"到输出中并以库的内容来代替它们。

比起在习题 21 中分析过的较简单方法,这个方法有多好?

28. [*25*] 正文中仅仅考虑了所有有待排序的记录都有一个固定大小的情况,对于可变长的记录,替代选择应如何做才好?

29. 考虑已经右穿线了的一个完全二叉树的 2^k 个节点,下面为 $k=3$ 时的图示:



(同 2.3.1-(10)做比较,顶上的节点是表头,而虚线是穿线链接。在本习题中,我们不关心排序,而是像在图 63 中一样,当在节点 1 上面加上类似表头的节点 0 时,关心完全二叉树的结构)。

说明怎样把一个大的失利者树的 2^{n+k} 个内部节点指定到 2^k 个宿主节点上,使得:(i)每个宿主节点恰保留大的树的 2^n 个节点;(ii)大树中相邻的节点或者被指定到相同的宿主节点上,或者被指定到相邻的(链接的)宿主节点;(iii)在大的树中没有两对相邻节点在宿主树中被同一个链所分开[因此在一个大的二叉树网络中,多个虚拟处理器可以被映射到实际的处理器上而不会造成通信链接中的过度拥挤]。

30. [*29*] 试证明,任何满足(i),(ii)和(iii)的 2^k-节点的宿主图在节点之间必然有至少 $2^k + 2^{k-1} - 1$ 条边(链接)。在这个意义下,如果 $n \geq k \geq 1$,则上题中的构造是最优的。

*5.4.2 多阶段合并

现在我们已经看到可以如何构造初始路段,我们将考虑各种型式,这些型式可被用来把这些路段分布到磁带上,并把它们合并在一起直到仅剩下一个路段为止。

从假定有 3 条磁带 T1,T2 和 T3 可资利用开始;在 5.4 节开始处所描述的"平衡合并"技术,可以对 $P=2$ 和 $T=3$ 使用,这时它采取如下的形式:

B1. 交替地在 T1 磁带和 T2 磁带上分布路段。

B2. 把 T1 和 T2 的路段合并到 T3 上;然后如果 T3 仅包含一个路段时便停止。

B3. 把 T3 的路段交替地拷贝到 T1 和 T2 上,然后返回到 B2。 ▌

如果初始分布扫描产生 S 个路段,则第一次合并扫描将在 T3 上产生 $\lceil S/2 \rceil$ 个路段,第二次合并扫描将产生 $\lceil S/4 \rceil$ 个,等等。于是,如果说 $17 \leq S \leq 32$,则我们将有 1 趟分配扫描,5 趟合并扫描,以及 4 趟拷贝扫描。一般地说,若 $S > 1$,对于所有数据的扫描次数是 $2\lceil \lg S \rceil$。

在此过程中的拷贝扫描是不需要的,因为它们并不减少路段数。如果我们使用一种两阶段的过程,则有一半的拷贝可以避免。

A1. 在磁带 T1 和 T2 上交替地分布初始路段。
A2. 把 T1 和 T2 上的路段合并到 T3 上；然后如果 T3 仅包含一个路段，则停止。
A3. 把 T3 上一半的路段拷贝到 T1 上。
A4. 把 T1 和 T3 的路段合并到 T2 上；然后如果 T2 仅含一个路段，则停止。
A5. 把 T2 上一半的路段拷贝到 T1，返回 A2。 ▌

对数据的扫描次数已经减少到 $\frac{3}{2}\lceil \lg S \rceil + \frac{1}{2}$，因为步骤 A3 和 A5 仅仅做"半次扫描"；因此节省了大约 25% 的时间。

如果我们从 T1 上的 F_n 个路段和 T2 上的 F_{n-1} 个路段开始，这里 F_n 和 F_{n-1} 是连续的斐波那契数，则实际上可以完全消去拷贝。例如，考虑 $n=7, S=F_n+F_{n-1}=13+8=21$ 的情况：

阶段	T1 的内容	T2 的内容	T3 的内容	注释
1	1,1,1,1,1,1,1,1,1,1,1,1,1	1,1,1,1,1,1,1,1		初始分布
2	1,1,1,1,1	—	2,2,2,2,2,2,2,2	把 8 个路段合并到 T3
3	—	3,3,3,3,3	2,2,2	把 5 个路段合并到 T2
4	5,5,5	3,3	—	把 3 个路段合并到 T1
5	5	—	8,8	把 2 个路段合并到 T3
6	—	13	8	把 1 个路段合并到 T2
7	21	—	—	把 1 个路段合并到 T1

例如，若把每个初始路段的相对长度定为 1，则"2,2,2,2,2,2,2,2"表示相对长度为 2 的 8 个路段，斐波那契数在这个图表中是无所不在的！

只有阶段 1 和 7 对数据进行了完全的扫描；阶段 2 仅仅处理初始路段的 16/21；阶段 3 仅处理 15/21，等等，因此如果我们假定初始路段都有近似相等的长度，则"扫描"总数就成为 $(21+16+15+15+16+13+21)/21=5\frac{4}{7}$。通过比较，以上的两阶段的过程将要求 8 次扫描以对 21 个初始的路段进行排序。我们将看到，一般地，这"斐波那契"型式近似地要求 $1.04\lg S+0.99$ 次扫描，使得它同一个 4-磁带平衡合并相匹敌，然而它只要求 3 条磁带。

同样的思想可以推广到 T 条磁带，对任何 $T\geqslant 3$，使用 $(T-1)$ 路合并。例如我们将看到，4-磁带的情形只要求对数据进行大约 $.703\lg S+0.96$ 次扫描。推广的形式涉及推广的斐波那契数。考虑下边 6-磁带的例子：

阶段	T1	T2	T3	T4	T5	T6	处理的初始路段
1	1^{31}	1^{30}	1^{28}	1^{24}	1^{16}	—	$31+20+28+24+16=129$
2	1^{15}	1^{14}	1^{12}	1^{8}	—	5^{16}	$16\times5=80$
3	1^{7}	1^{6}	1^{4}	—	9^{8}	5^{8}	$8\times9=72$
4	1^{3}	1^{2}	—	17^{4}	9^{4}	5^{4}	$4\times17=68$
5	1^{1}	—	33^{2}	17^{2}	9^{2}	5^{2}	$2\times33=66$
6	—	65^{1}	33^{1}	17^{1}	9^{1}	5^{1}	$1\times65=65$
7	129^{1}	—	—	—	—	—	$1\times129=129$

这里 1^{31}代表相对长度为 1 的 31 个路段，等等；这里已经从头到尾使用了五路合并。这一般的型式是由 R. L. Gilstad 提出的[*Proc. Eastern Joint. Computer Conf* . **18** (1960), 143～148]。他把它称做多阶段合并。3-磁带的情况已经较早地为 B. K. Betz 发现[unpublished memorandum, Minneapolis-Honeywell Regulator Co. (1956)]。

为了如同上边的例子中那样进行多阶段的合并工作，我们在每个阶段之后，需要使诸路段在磁带上构成"完全的斐波那契分布"。通过由底向上读上面的表，可以看到当 $T=6$ 时，前 7 个完全的斐波那契分布是{1,0,0,0,0}，{1,1,1,1,1}，{2,2,2,2,1}，{4,4,4,3,2}，{8,8,7,6,4}，{16,15,14,12,8}和{31,30,28,24,16}，我们面临的较大问题是：

1. 这些完全的斐波那契分布所基于的规则是什么？
2. 如果 S 不对应于一个完全的斐波那契分布，则我们怎么办？
3. 我们应该怎样设计初始的分布扫描，使得它在磁带上产生所希望的配置？
4. 作为 S(初始路段个数)的函数，一个 T-磁带多阶段合并要求对数据进行多少"次"扫描？

我们将依次地讨论这 4 个问题，首先给出"容易的答案"而后进行更加详细的分析。

完全的斐波那契分布可以通过周期地转动磁带的内容来向后运行而得到。例如，当 $T=6$ 时，我们有下列的路段分布：

级	T1	T2	T3	T4	T5	总数	最后的输出将在
0	1	0	0	0	0	1	T1
1	1	1	1	1	1	5	T6
2	2	2	2	2	1	9	T5
3	4	4	4	3	2	17	T4
4	8	8	7	6	4	33	T3
5	16	15	14	12	8	65	T2
6	31	30	28	24	16	129	T1
7	61	59	55	47	31	253	T6
8	120	116	108	92	61	497	T5
……							
n	a_n	b_n	c_n	d_n	e_n	t_n	$T(k)$
$n+1$	a_n+b_n	a_n+c_n	a_n+d_n	a_n+e_n	a_n	t_n+4a_n	$T(k-1)$
……							

(1)

在初始的分布之后，磁带 T6 将总是空的。

从级 n 到级 $n+1$ 进行的规则表明，在每一级中都有条件

$$a_n \geqslant b_n \geqslant c_n \geqslant d_n \geqslant e_n \tag{2}$$

成立。事实上，从(1)容易看出

$$\begin{aligned} e_n &= a_{n-1} \\ d_n &= a_{n-1} + e_{n-1} = a_{n-1} + a_{n-2} \\ c_n &= a_{n-1} + d_{n-1} = a_{n-1} + a_{n-2} + a_{n-3} \end{aligned}$$

$$\begin{aligned} b_n &= a_{n-1} + c_{n-1} = a_{n-1} + a_{n-2} + a_{n-3} + a_{n-4} \\ a_n &= a_{n-1} + b_{n-1} = a_{n-1} + a_{n-2} + a_{n-3} + a_{n-4} + a_{n-5} \end{aligned} \tag{3}$$

其中 $a_0=1$,而且对于 $n=-1,-2,-3,-4$ 我们令 $a_n=0$。

第 p 阶斐波那契数 $F_n^{(p)}$ 定义为

$$\begin{aligned} &F_n^{(p)} = F_{n-1}^{(p)} + F_{n-2}^{(p)} + \cdots + F_{n-p}^{(p)} \quad \text{对于 } n \geqslant p \\ &F_n^{(p)} = 0 \qquad \text{对于 } 0 \leqslant n \leqslant p-2 \\ &F_{p-1}^{(p)} = 1 \end{aligned} \tag{4}$$

换句话说,我们开始有 $p-1$ 个 0,然后是 1,然后每个数是前边 p 个值之和。当 $p=2$ 时,这是通常的斐波那契数列。对于更大的 p 值,这个序列似乎首先是由 V. Schlegel 在 *El Progreso Matemático* **4**(1894),173~174 上加以研究的。Schlegel 导出了生成函数

$$\sum_{n\geqslant 0} F_n^{(p)} z^n = \frac{z^{p-1}}{1-z-z^2-\cdots-z^p} = \frac{z^{p-1}-z^p}{1-2z+z^{p+1}} \tag{5}$$

(3)中最后的等式表明在一个 6-磁带多阶段合并期间,在 T1 上的路段数是第 5 阶斐波那契数 $a_n=F_{n+4}^{(5)}$。

一般地说,如果我们置 $P=T-1$,则 T 条磁带的多阶段合并分布将以同样方式对应于第 P 阶斐波那契数。对于 $1\leqslant k\leqslant P$,第 k 条磁带在完全的第 n 级分布中得到

$$F_{n+p-2}^{(P)} + F_{n+p-3}^{(P)} + \cdots + F_{n+k-2}^{(P)}$$

个初始路段,因此在所有磁带上的初始路段总数为

$$t_n = PF_{n+p-2}^{(P)} + (P-1)F_{n+P-3}^{(P)} + \cdots + F_{n-1}^{(P)} \tag{6}$$

这样就解决了“完全的斐波那契分布”的问题。但如果对于任意 n,S 不恰巧等于 t_n,则我们应做什么呢?而且开头在这些磁带上我们怎样得到路段呢?

当 S 不是完全的(有少数值是这样的)时候,我们还可以像在平衡的 P 路合并中那样做,加上人工的“虚拟路段”后使得可以假想 S 仍然是完全的。有若干种方式来附加虚拟路段,我们尚不准备分析这样做的“最好”的方式。我们将首先讨论分布的方法和指定虚拟路段的方法,它并非严格地是最优的,尽管它有简便的长处而且似乎比所有其它同样简单的方法都要好些。

算法 D(具有“水平”分布的多阶段合并排序) 这个算法把初始路段疏散到磁带上,一次一个路段,直到初始路段的供给穷尽为止。然后它确定这些磁带如何被合并。假定有 $T=P+1\geqslant 3$ 台可利用的磁带机,同时使用 P 路合并。磁带 T 可以用来保存输入,因为它不接收任何初始的路段。维护如下一些表项:

A[j],$1\leqslant j\leqslant T$:我们正在力求的完全斐波那契分布。

D[j],$1\leqslant j\leqslant T$:假定应出现在设备编号为 j 的逻辑磁带开始处的虚拟路段个

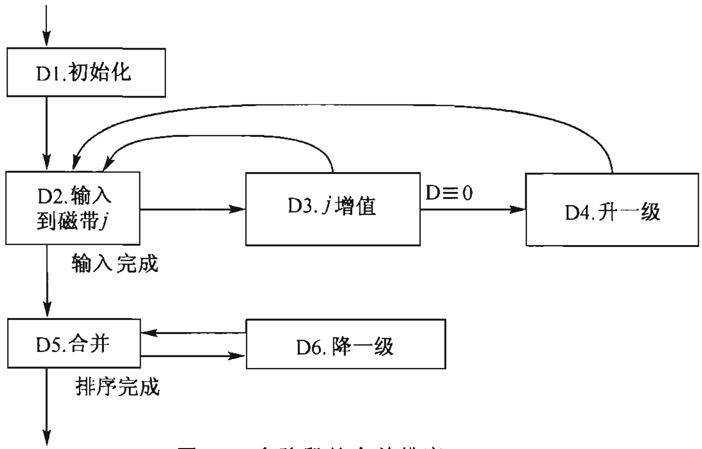


图 68 多阶段的合并排序

数。

TAPE[j],$1\leqslant j\leqslant T$:对应于逻辑磁带设备号 j 的物理磁带设备号(处理“逻辑磁带设备号”是方便的。对它分配物理磁带设备号是随算法的进行而变化的)。

D1.[初始化] 对于 $1\leqslant j<T$,置 A[j]←D[j]−1 和 TAPE[k]←j。置 A[T]←D[T]←0 和 TAPE[T]←T。然后置 l←1,j←1。

D2.[输入到磁带 j] 写一个路段到编号为 j 的磁带上,并且 D[j]减 1。然后如果输入已穷尽,则重绕所有的磁带并转到步骤 D5。

D3.[j 增值] 如果 D[j]<D[$j+1$],则 j 增加 1,并且返回到 D2.否则如果 D[j]=0,则转到 D4。否则置 j←1,并且返回到 D2。

D4.[升一级] 置 $l\leftarrow l+1$,$a\leftarrow A[1]$,然后对于 $j=1,2,\cdots,P$(在这个次序下)置 D[j]←a+A[$j+1$]−A[j]和 A[j]←a+A[$j+1$](见(1)并注意 A[$P+1$]总是 0。这时我们将有 D[1]⩾D[2]⩾…⩾D[T])。现在置 j←1 并且返回到 D2。

D5.[合并] 如果 $l=0$,则排序完成而且输出在 TAPE[1]上。否则,把 TAPE[1],…,TAPE[P]的路段合并到 TAPE[T]上,直到 TAPE[P]成为空的而且 D[P]=0 为止。对于每一个被合并的路段,合并过程应该如下操作:如果对所有 j,$1\leqslant j\leqslant P$,D[j]>0,则 D[T]增加 1 且对于 $1\leqslant j\leqslant P$,每个 D[j]减 1;否则从每一个使得 D[j]=0 的 TAPE[j]合并一个路段,而且对每个其它的 j,D[j]减 1(于是虚拟路段被想像为在磁带的开始而不是在末尾)。

D6.[降一级] 置 $l\leftarrow l-1$。重绕 TAPE[P]和 TAPE[T](实际上 TAPE[P]的重绕在步骤 D5 中,在输入了它的最后一组路段之后就可以开始)。然后置(TAPE[1],TAPE[2],…,TAPE[T])←(TAPE[T],TAPE[1],…,TAPE[$T-1$]),(D[1],D[2],…,D[T])←(D[T],D[1],…,D[$T-1$]),并返回到步骤 D5。▌

此算法的步骤 D3 中如此简洁地叙述的分布规则，是打算尽可能在每条磁带上设置相等个数的虚拟路段。图 69 所示为我们在一个 6-磁带的排序中从级 4(33 个路段)进行到级 5(65 个路段)时分布的次序；如果仅仅有，比如说，53 个初始的路段，则所有编号为 54 和更高的路段都将被处理作虚拟的(这些路段实际上被写到这条磁带的末尾，但是最好把它们想像成写到开始处，因为已经假定虚拟路段都是在开始处)。

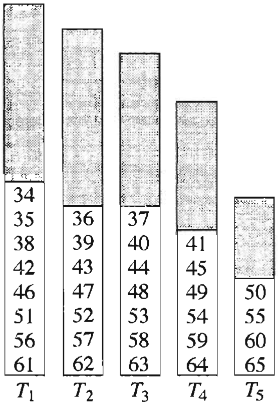


图 69 当从级 4 进行到级 5 时，路段 34 到 65 被分布于各磁带上的次序(见(1)的完全分布表)。阴影区域表示当到达级 4 时已被分布好的前 33 个路段

我们现在已经讨论了上边列出的前 3 个问题，剩下要考虑的是数据“扫描”的次数。把 6-磁带的例子同表(1)进行比较，我们看到，当 $S = t_6$ 时，处理的初始路段的总数是 $a_5t_1 + a_4t_2 + a_3t_3 + a_2t_4 + a_1t_5 + a_0t_6$，初始的分布扫描不计在内。习题 4 导出生成函数

$$a(z) = \sum_{n\geqslant 0} a_n z^n = \frac{1}{1 - z - z^2 - z^3 - z^4 - z^5}$$
$$t(z) = \sum_{n\geqslant 1} t_n z^n = \frac{5z + 4z^2 + 3z^3 + 2z^4 + z^5}{1 - z - z^2 - z^3 - z^4 - z^5} \tag{7}$$

由此得出，一般说来当 $S = t_n$ 时处理的初始路段数准确地等于在 $a(z)t(z)$ 中 z^n 的系数，加上 t_n(对于初始的分布扫描)。这使得有可能如习题 5～7 所示来计算多阶段合并的渐近行为，并得到表 1 中所示的结果。

表 1 多阶段的合并排序的近似特性

磁带	阶段	扫描	扫描/阶段	增长率
3	$2.078 \ln S + 0.672$	$1.504 \ln S + 0.992$	72%	1.6180340
4	$1.641 \ln S + 0.364$	$1.015 \ln S + 0.965$	62%	1.8392868
5	$1.524 \ln S + 0.078$	$0.863 \ln S + 0.921$	57%	1.9275620
6	$1.479 \ln S - 0.185$	$0.795 \ln S + 0.864$	54%	1.9659482
7	$1.460 \ln S - 0.424$	$0.762 \ln S + 0.797$	52%	1.9835828
8	$1.451 \ln S - 0.642$	$0.744 \ln S + 0.723$	51%	1.9919642
9	$1.447 \ln S - 0.838$	$0.734 \ln S + 0.646$	51%	1.9960312
10	$1.445 \ln S - 1.017$	$0.728 \ln S + 0.568$	50%	1.9980295
20	$1.443 \ln S - 2.170$	$0.721 \ln S - 0.030$	50%	1.9999981

在表 1 中,"增长率"为 $\lim_{n\to\infty} t_{n+1}/t_n$,此即路段的个数在每一级增长的近似因子。"扫描"表示每个记录被处理的平均次数,即 $1/S$ 乘上在分布和合并阶段处理的初始路段的总数。对于完全分布来说,当 $S\to\infty$ 时,存在一个 $\varepsilon>0$,使得在每种情况下所述的扫描和阶段数都正确到 $O(S^{-\varepsilon})$。

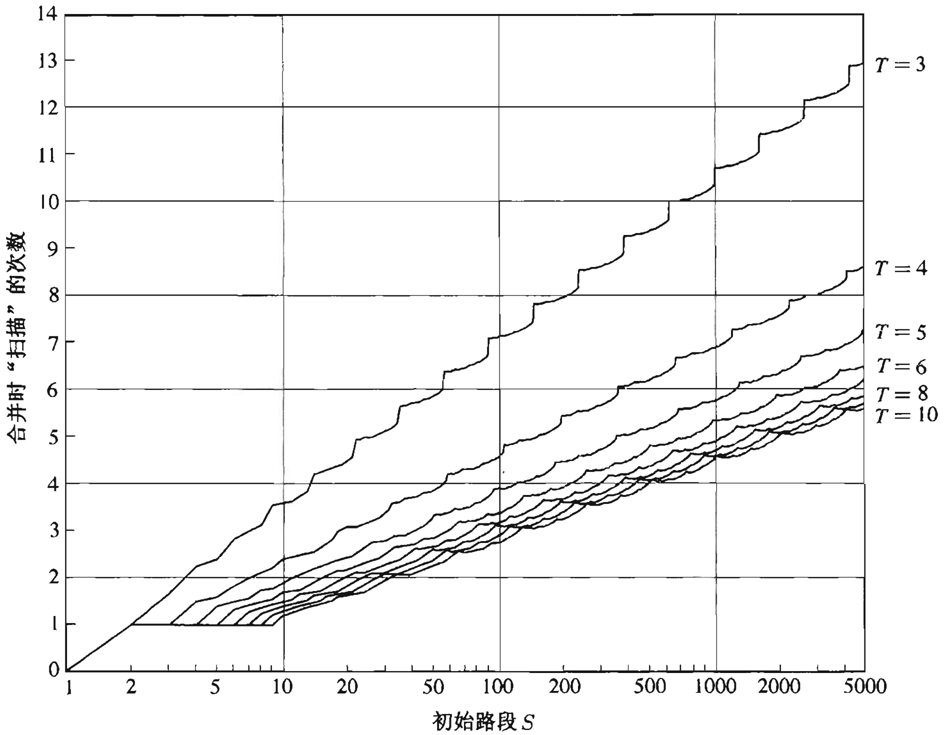


图 70 使用算法 D 的多阶段合并的效率

图 70 所示为用算法 D 处理不完全数的情况时,作为 S 的一个函数,每个记录被合并的平均次数。注意,对于 3 条磁带,恰恰在完全分布之后就出现了相对低效率的"峰值",但当有 4 条或更多的磁带时,这种现象就很少见了。使用 8 条或更多的磁带对 6 条或 7 条磁带的改进相当小。

更仔细的考察 在要求 k 次扫描的一个平衡的合并中,在排序过程中每个记录恰巧被处理 k 次。但是多阶段的过程却并非这样;某些记录可能得到比其它记录多许多次的处理,而且同时,如果把虚拟路段安排到经常被处理的位置,则我们就能赢得速度。

因此让我们更仔细地研究多阶段的分布,不像在(1)中那样仅仅考察在每个磁带上的路段数,我们把每一路段同它的合并数,即在整个多阶段排序期间将被处理的次数,结合起来,就得到下列的表,以代替(1):

级	T1	T2	T3	T4	T5	
0	0	—	—	—	—	
1	1	1	1	1	1	
2	21	21	21	21	2	
3	3221	3221	3221	322	32	
4	43323221	43323221	4332322	433232	4332	
5	5443433243323221	544343324332322	54434332433232	544343324332	54434332	
……						
n	A_n	B_n	C_n	D_n	E_n	
$n+1$	$(A_n+1)B_n$	$(A_n+1)C_n$	$(A_n+1)D_n$	$(A_n+1)E_n$	A_n+1	(8)
……						

如果我们从 n 级分布开始，A_n 是 a_n 个值的串，表示 T1 上每个路段的合并数；B_n 是对于 T2 的对应的串，等等。记号“$(A_n+1)B_n$”意味着“A_n 中的所有值增加 1，后边紧接以 B_n”。

图 71(a)所示为出现在末端的 A_5, B_5, C_5, D_5, E_5，示出每个路段的合并数如何出现在磁带上；注意，例如，在每条磁带开始的路段将被处理 5 次，而在 T1 末尾的路段将仅仅被处理 1 次。多阶段合并的这种区别对待的情况，使得把一个虚拟路段放置在磁带的开始比放在磁带的末尾要好得多。图 71(b)所示为在 5 级多阶段合并中路段分布的一个最佳次序，把每个新路段放置到具有最小合并次数的位置上。算法 D(见图 69)并不是十分好的，因为在所有“3”的位置被用完之前，它填充了某些“4”的位置。

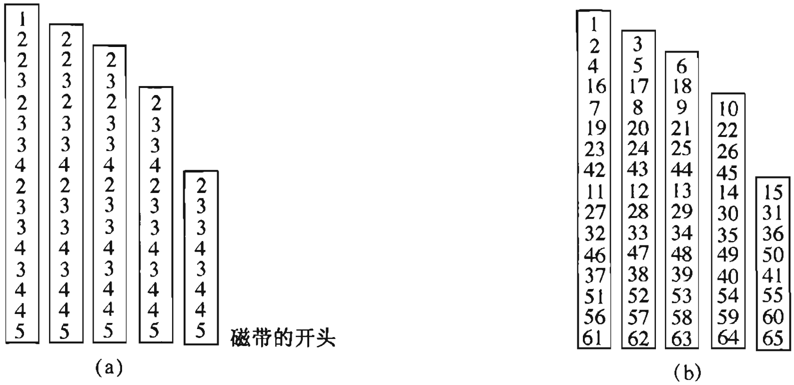


图 71　对于 6 条磁带的第 5 级多阶段的分布的分析

(a) 合并数；(b) 最优分布的次序。

递推关系(8)表明，B_n, C_n, D_n, E_n 的每一个都是 A_n 的初始子串。事实上，我们可以使用(8)来推导公式

$$\begin{aligned} E_n &= (A_{n-1})+1 \\ D_n &= (A_{n-1}A_{n-2})+1 \\ C_n &= (A_{n-1}A_{n-2}A_{n-3})+1 \\ B_n &= (A_{n-1}A_{n-2}A_{n-3}A_{n-4})+1 \\ A_n &= (A_{n-1}A_{n-2}A_{n-3}A_{n-4}A_{n-5})+1 \end{aligned} \tag{9}$$

这推广了仅仅处理这些串的长度的公式(3)。进而，从定义 A 的规则可以看出，在每级的开头出现的实际上是相同的结构；我们有

$$A_n = n - Q_n \tag{10}$$

其中 Q_n 是由法则

$$\begin{aligned} &Q_n = Q_{n-1}(Q_{n-2}+1)(Q_{n-3}+2)(Q_{n-4}+3)(Q_{n-5}+4) \quad \text{对于 } n \geqslant 1 \\ &Q_0 = 0, \\ &Q_n = \epsilon \quad (\text{空串}) \quad \text{对于 } n < 0 \end{aligned} \tag{11}$$

定义的 a_n 个值的一个串。由于 Q_n 以 Q_{n-1} 开始，我们可以考察无穷的串 Q_∞；它的前 a_n 个元素等于 Q_n，这个串 Q_∞ 实际上表征了在多阶段分布中所有的合并数。在 6-磁带的情况下，

$$\begin{aligned} Q_\infty = &0112122312232334122323342334344 12 \\ &23233423343445233434453444545 12232\cdots \end{aligned} \tag{12}$$

习题 11 包含了对于这个串的一种有趣的解释。

设 A_n 是串 $m_1m_2\cdots m_{a_n}$，令

$$A_n(x) = x^{m_1} + x^{m_2} + \cdots + x^{m_{a_n}}$$

是相应的计算每个合并数出现次数的生成函数；而且类似地定义 $B_n(x), C_n(x), D_n(x), E_n(x)$。例如，$A_4(x) = x^4 + x^3 + x^3 + x^2 + x^3 + x^2 + x^2 + x = x^4 + 3x^3 + 3x^2 + x$。关系(9)告诉我们，对于 $n \geqslant 1$，有

$$\begin{aligned} E_n(x) &= x(A_{n-1}(x)) \\ D_n(x) &= x(A_{n-1}(x) + A_{n-2}(x)) \\ C_n(x) &= x(A_{n-1}(x) + A_{n-2}(x) + A_{n-3}(x)) \\ B_n(x) &= x(A_{n-1}(x) + A_{n-2}(x) + A_{n-3}(x) + A_{n-4}(x)) \\ A_n(x) &= x(A_{n-1}(x) + A_{n-2}(x) + A_{n-3}(x) + A_{n-4}(x) + A_{n-5}(x)) \end{aligned} \tag{13}$$

其中 $A_0(x)=1$，而且对于 $n=-1,-2,-3,-4$，$A_n(x)=0$。因此

$$\sum_{n\geqslant 0} A_n(x)z^n = \frac{1}{1 - x(z+z^2+z^3+z^4+z^5)} = \sum_{k\geqslant 0} x^k(z+z^2+z^3+z^4+z^5)^k \tag{14}$$

考虑所有磁带上的路段，我们设

$$T_n(x) = A_n(x) + B_n(x) + C_n(x) + D_n(x) + E_n(x) \quad n \geqslant 1 \tag{15}$$

由式(13)我们立即有

$$T_n(x) = 5A_{n-1}(x) + 4A_{n-2}(x) + 3A_{n-3}(x) + 2A_{n-4}(x) + A_{n-5}(x)$$

因此

$$\sum_{n \geqslant 1} T_n(x) z^n = \frac{x(5z + 4z^2 + 3z^3 + 2z^4 + z^5)}{1 - x(z + z^2 + z^3 + z^4 + z^5)} \tag{16}$$

式(16)的形式说明,容易计算出 $T_n(x)$ 的系数:

	z	z^2	z^3	z^4	z^5	z^6	z^7	z^8	z^9	z^{10}	z^{11}	z^{12}	z^{13}	z^{14}
x	5	4	3	2	1	0	0	0	0	0	0	0	0	0
x^2	0	5	9	12	14	15	10	6	3	1	0	0	0	0
x^3	0	0	5	14	26	40	55	60	57	48	35	20	10	4
x^4	0	0	0	5	19	45	85	140	195	238	260	255	220	170
x^5	0	0	0	0	5	24	69	154	294	484	703	918	1088	1168

(17)

这个表的列给出了 $T_n(x)$;例如,$T_4(x) = 2x + 12x^2 + 14x^3 + 5x^4$。在第一行之后,这个表中的每个项目都是位于前边一行中对应项目左边的 5 个项目之和。

在一个"完全的"第 n 级分布中,路段的数目是 $T_n(1)$,而且当这些路段被合并时处理的总数是微商 $T_n'(1)$,现在

$$\sum_{n \geqslant 1} T_n'(x) z^n = \frac{5z + 4z^2 + 3z^3 + 2z^4 + z^5}{(1 - x(z + z^2 + z^3 + z^4 + z^5))^2} \tag{18}$$

在式(16)和式(18)中置 $x = 1$,就可得出一个同我们前面的结论相一致的结果,这个结论就是:对于一个完全的第 n 级分布,合并的处理是 $a(z)t(z)$ 中 z^n 的系数;参见式(7)。

我们可以使用函数 $T_n(x)$ 来确定,当以最优方式加上虚拟路段时所涉及的工作量有多少。令 $\sum_n(m)$ 是在第 n 级分布中最小的 m 个合并数之和。通过考察式(17)的诸列,很容易计算出这些值来,而且我们发现 $\sum_n(m)$ 由下表给出:

	$m=1$	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
$n=1$	1	2	3	4	5	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
$n=2$	1	2	3	4	6	8	10	12	14	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
$n=3$	1	2	3	5	7	9	11	13	15	17	19	21	24	27	30	33	36	∞	∞	∞	∞
$n=4$	1	2	4	6	8	10	12	14	16	18	20	22	24	26	29	32	35	38	41	44	47
$n=5$	1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	32	35	38	41	44	47
$n=6$	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	33	36	39	42	45	48
$n=7$	2	4	6	8	10	12	14	16	18	20	23	26	29	32	35	38	41	44	47	50	53

(19)

例如,如果我们希望用一个第 3 级分布来对 17 个路段排序,则处理的总数量为 $\sum_3$

(17) = 36;但是如果我们使用第 4 级或第 5 级分布,以及最优地放置虚拟路段的位置,则在合并阶段的处理总数量就只有$\sum_4(17)=\sum_5(17)=35$。使用第 4 级是更好的,尽管 17 对应于一个"完全的"第 3 级分布! 其实,当 S 变得很大时,已经证实级的最优个数要比在算法 D 中所使用的多得多。

习题 14 表明有一个非减的数序列 M_n,使得级 n 对于$M_n\leqslant S<M_{n+1}$是"最优"的,但对于 $S\geqslant M_{n+1}$则不然。在 6-磁带的情况下,我们刚刚计算的$\sum_n(m)$的表说明

$$M_0=0,\ M_1=2,\ M_2=6,\ M_3=10,\ M_4=14$$

上边的讨论仅仅处理了 6-磁带的情况,但是显然,同样的思想可应用于对于任何 $T\geqslant 3$ 的 T 条磁带的多阶段的合并;在所有适当的位置中,我们都简单地以 $P=T-1$ 代替 5。表 2 为对于各种 T 值所得到的序列M_n。表 3 和图 72 指出了在做成虚拟路段的一个最优分布之后,被处理的初始路段总数(在表 3 的下部出现的公式应当有保留地采用,因为它们是在变程 $1\leqslant S\leqslant 5000$(或者对于 $T=3$ 为 $1\leqslant S\leqslant 10000$)上"最小平方符合"的;这导致了有些反常的特性,因为给定的 S 的范围并非对于所有的 T 都是同样有利的。当 $S\to\infty$时,在一个最优的多阶段分布之后处理的初始路段数渐近于 $S\log_P S$,但是收敛于此渐近极限值的过程是极端缓慢的)。

表 2 对应于最优级的路段数

线路	$T=3$	$T=4$	$T=5$	$T=6$	$T=7$	$T=8$	$T=9$	$T=10$	
1	2	2	2	2	2	2	2	2	M_1
2	3	4	5	6	7	8	9	10	M_2
3	4	6	8	10	12	14	16	18	M_3
4	6	10	14	14	17	20	23	26	M_4
5	9	18	23	29	20	24	28	32	M_5
6	14	32	35	43	53	27	32	37	M_6
7	22	55	76	61	73	88	35	41	M_7
8	35	96	109	154	98	115	136	44	M_8
9	56	173	244	216	283	148	171	199	M_9
10	90	280	359	269	386	168	213	243	M_{10}
11	145	535	456	779	481	640	240	295	M_{11}
12	234	820	1197	1034	555	792	1002	330	M_{12}
13	378	1635	1563	1249	1996	922	1228	1499	M_{13}
14	611	2401	4034	3910	2486	1017	1432	1818	M_{14}
15	988	4959	5379	4970	2901	4397	1598	2116	M_{15}
16	1598	7029	6456	5841	10578	5251	1713	2374	M_{16}
17	2574	14953	18561	19409	13097	5979	8683	2576	M_{17}
18	3955	20583	22876	23918	15336	6499	10069	2709	M_{18}
19	6528	44899	64189	27557	17029	30164	11259	15787	M_{19}

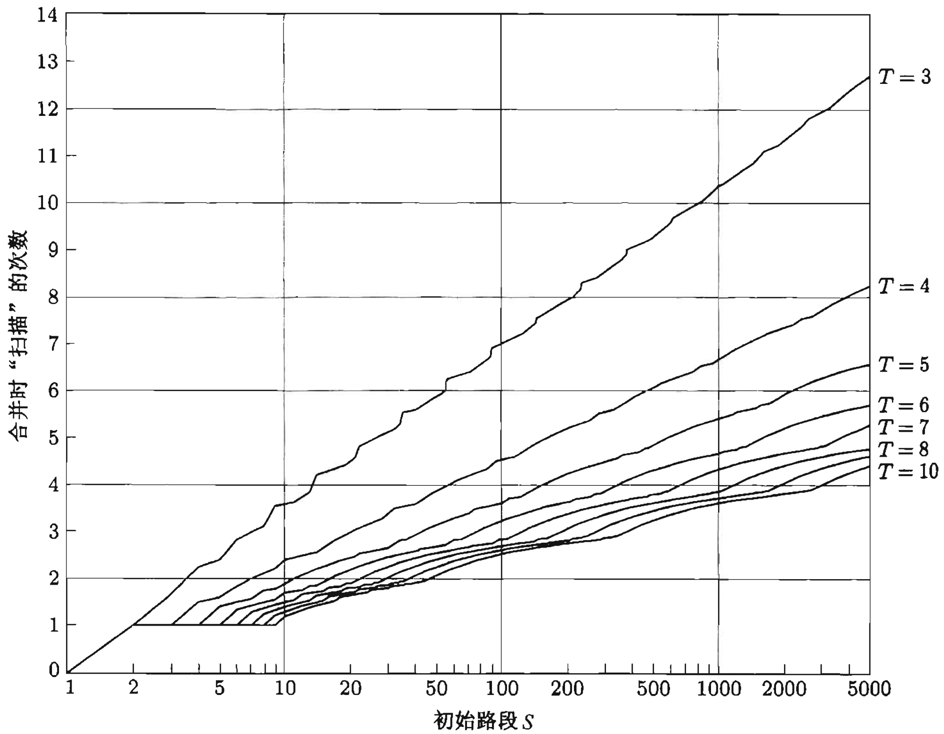


图 72　使用和图 70 同样的假定，具有最优初始分布的多阶段合并的效率

表 3　在最优的多阶段合并期间处理的初始路段

S	T = 3	T = 4	T = 5	T = 6	T = 7	T = 8	T = 9	T = 10	
10	36	24	19	17	15	14	13	12	
20	90	60	49	44	38	36	34	33	
50	294	194	158	135	128	121	113	104	
100	702	454	362	325	285	271	263	254	
500	4641	3041	2430	2163	1904	1816	1734	1632	
1000	10371	6680	5430	4672	4347	3872	3739	3632	
5000	63578	41286	32905	28620	26426	23880	23114	22073	
S	(1.51	0.951	0.761	0.656	0.589	0.548	0.539	0.488)	$\times S \ln S$
	(−.11	+.14	+.16	+.19	+.21	+.20	+.02	+.18)	$\times S$

表 4 表明算法 D 的分布方法堪与表 3 的最优分布结果相匹敌。显然，当 S 和 T 变得很大时，算法 D 并不是非常接近最优的；但是还不清楚在这两种情况下如何能比算法 D 做得更好而又不致搞得过分复杂，特别是，如果我们事先并不知道 S 的情

况时就更是如此。幸而我们很少碰上很大的 S(见 5.4.6 小节),所以算法 D 实际上并不是太坏的;事实上,它相当好。

多阶段排序首先是由 W. C. Carter 从数学上加以分析的[*Proc. IFIP Congress* (1962),62～66]。关于最优虚拟路段安排的以上许多结果,原来是由 B. Sackman 和 T. Singer 给出的["A vector model for merge sort analysis",an unpublished paper presented at the ACM Sort Symposium(November 1962),21 pages]。Sackman 后来提出在算法 D 中使用的分布的水平方法。D. Donald Shell[*CACM* **14**(1971),713～719;**15**(1972),28]独立地发展了这个理论,指出了关系(10),并对于若干不同的分布算法进行了详细的研究。Derek A. Zave[*SICOMP* **6**(1977),1～39]已经做出了进一步的有教益的发展和改进;习题 15 到 17 讨论了 Zave 的某些结果。生成函数(16)首先是由 W. Burge 研究的[*Proc. IFIP Congress* (1971),**1**,454～459]。

表 4　在标准多阶段合并时处理的初始路段

S	$T=3$	$T=4$	$T=5$	$T=6$	$T=7$	$T=8$	$T=9$	$T=10$
10	36	24	19	17	15	14	13	12
20	90	62	49	44	41	37	34	33
50	294	194	167	143	134	131	120	114
100	714	459	393	339	319	312	292	277
500	4708	3114	2599	2416	2191	2100	2047	2025
1000	10730	6920	5774	5370	4913	4716	4597	4552
5000	64740	43210	36497	32781	31442	29533	28817	28080

重绕时间为何?　迄今,我们都以"被处理的初始路段"作为比较磁带合并策略的有效性的惟一测度。但本节开始的例子中在阶段 2 到 6 的每一阶段之后,计算机都必须等候两条磁带重绕;在下一阶段可以进行之前,先前的输出磁带和新的当前输出磁带两者都必须重新定位到开头处。这会引起一个相当大的迟延,因为先前的输出磁带一般都包含了相当百分比的正被排序的记录(见表 1 的"扫描/阶段"列)。在所有这些重绕操作期间,让计算机闲得无聊是令人遗憾的,但如果我们使用一种不同的合并形式,那就可以用其它磁带来完成有用的工作。

简单地修改多阶段的过程,就能解决这个问题,尽管它至少要用 5 条磁带[见 Y. Césari, Thesis, U. of Paris(1968),25～27,其中把这个思想归功于 J. Caron]。在 Caron 的方案里,在每个阶段,都把 T－3 条磁带上的路段合并到另一条磁带上,而剩下的两条磁带重绕。

例如,考虑 6 条磁带和 49 个初始路段的情况。在下表中,R 表示在这个阶段进行重绕,而且假定 T5 包含原来的输入:

阶段	T1	T2	T3	T4	T5	T6	写的时间	重绕时间
1	1^{11}	1^{17}	1^{13}	1^{8}	—	(R)	49	17

2	(R)	1^9	1^5	—	R	3^8	$8\times3=24$	$49-17=32$
3	1^6	1^4	—	R	3^5	R	$5\times3=15$	max(8,24)
4	1^2	—	R	5^4	R	3^4	$4\times5=20$	max(13,15)
5	—	R	7^2	R	3^3	3^2	$2\times7=14$	max(17,20)
6	R	11^2	R	5^2	3^1	—	$2\times11=22$	max(11,14)
7	15^1	R	7^1	5^1	—	R	$1\times15=15$	max(22,24)
8	R	11^1	7^0	—	R	23^1	$1\times23=23$	max(15,15)
9	15^1	11^1	—	R	33^0	R	$0\times33=0$	max(20,23)
10	(15^0)	—	R	49^1	(R)	(23^0)	$1\times49=49$	14

这里所有的重绕时间实际上都是重叠的,只有阶段 9(这是一个“虚拟阶段”,它为最后的合并做准备)以及在初始分布阶段之后(当所有磁带都被重绕时)除外。如果 t 是合并一个初始路段中全体记录的时间,且如果 r 是重绕一个初始路段的时间,则这个过程花费大约 $182t+40r$ 加上初始分布和最后重绕花费的时间。对于使用算法 D 的标准多阶段法,对应数字是 $140t+104r$,当 $r=\frac{3}{4}t$ 时它要差一些,而当 $r=\frac{1}{2}t$ 时则稍微好些。

我们对于标准多阶段法已说过的每件事情,都可适用于 Caron 的多阶段法;例如,序列 a_n 现在满足的不是(3)而是递推式

$$a_n = a_{n-2} + a_{n-3} + a_{n-4} \tag{20}$$

读者将发现,像我们分析标准的多阶段法那样来分析这个方法是有教益的,因为它将增进对于这两种方法的理解(比如,见习题 19 和 20)。

表 5 给出了关于 Caron 多阶段法的一些事实,它们类似于表 1 中的通常多阶段法的相应事实。注意,Caron 的方法就处理的路段数以及重绕的时间而言,实际上比对 8 条或更多磁带的多阶段法优越,尽管它进行了 $T-3$ 路合并而不是 $T-1$ 路合并!

表 5 Caron 多阶段合并排序的近似行为

磁带	阶　段	扫　描	扫描/阶段	增　长　率
5	$3.556\ln S+0.158$	$1.463\ln S+1.016$	41%	1.3247180
6	$2.616\ln S-0.166$	$0.951\ln S+1.014$	36%	1.4655712
7	$2.337\ln S-0.472$	$0.781\ln S+1.001$	33%	1.5341577
8	$2.216\ln S-0.762$	$0.699\ln S+0.980$	32%	1.5701473
9	$2.156\ln S-1.034$	$0.654\ln S+0.954$	30%	1.5900054
10	$2.124\ln S-1.290$	$0.626\ln S+0.922$	29%	1.6013473
20	$2.078\ln S-3.093$	$0.575\ln S+0.524$	28%	1.6179086

一个高阶的合并并不必然意味着一个有效的排序,在我们认识到这一点之前,

上述事实似乎是一种怪事。作为一个极端的例子,考虑把一个路段放到 T1 上而且把 n 个路段放到 T2,T3,T4,T5 上;如果我们交替地进行五路合并到 T6 和 T1 上,直到 T2,T3,T4,T5 变空为止,则处理时间是$(2n^2+3n)$个初始路段长度,实际上和 S^2 而不是和 $S \log S$ 成比例,尽管自始至终地进行了五路合并。

分带 重绕时间的有效重叠是在许多应用中所产生的一个问题,而不只限于排序中,而且有一个通常可资利用的一般方法。考虑一个迭代过程,它以如下方式使用两条磁带:

	T1	T2
阶段 1	输出 1	—
	重绕	—
阶段 2	输入 1	输出 2
	重绕	重绕
阶段 3	输出 3	输入 2
	重绕	重绕
阶段 4	输入 3	输出 4
	重绕	重绕

等等,其中“输出 k”意味着写出第 k 个输出文件,而“输入 k”就意味着读入它。当使用 3 条磁带时,如同 C. Weisert 所建议的[*CACM* **5** (1962),102]那样,可以免去重绕的时间:

	T1	T2	T3
阶段 1	输出 1.1	—	—
	输出 1.2	—	—
	重绕	输出 1.3	—
阶段 2	输入 1.1	输出 2.1	—
	输入 1.2	重绕	输出 2.2
	重绕	输入 1.3	输出 2.3
阶段 3	输出 3.1	输入 2.1	重绕
	输出 3.2	重绕	输入 2.2
	重绕	输出 3.3	输入 2.3
阶段 4	输入 3.1	输出 4.1	重绕
	输入 3.2	重绕	输出 4.2
	重绕	输入 3.3	输出 4.3

等等,这里“输出 $k.j$”意味着,写出第 k 个输出文件的第j 个 1/3,而“输入 $k.j$”就意味着读入它。实际上,如果重绕速度至少是读写速度的两倍,则所有的重绕时间都将被消去。这样的一个过程,其中每个阶段的输出都被分到一些磁带上,称为“分带”。

R. L. McAllester[*CACM* **7** (1964),158~159]已经证明,分带是一个把多阶段合并中的重绕时间加以重叠的有效方法。他的方法可以用于 4 条或更多的磁带,而且

它进行 $T-2$ 路合并。

再次假定有 6 条磁带，让我们来设计一个合并型式，它的操作如下：分开在每级上的输出，其中“I”，“O”，和“R”分别表示输入、输出和重绕。

级	T1	T2	T3	T4	T5	T6	输出路段的个数
7	I	I	I	I	R	O	u_7
	I	I	I	I	O	R	v_7
6	I	I	I	R	O	I	u_6
	I	I	I	O	R	I	v_6
5	I	I	R	O	I	I	u_5
	I	I	O	R	I	I	v_5
4	I	R	O	I	I	I	u_4
	I	O	R	I	I	I	v_4
3	R	O	I	I	I	I	u_3
	O	R	I	I	I	I	v_3
2	O	I	I	I	I	R	u_2
	R	I	I	I	I	O	v_2
1	I	I	I	I	R	O	u_1
	I	I	I	I	O	R	v_1
0	I	I	I	R	O	I	u_0
	I	I	I	O	R	I	v_0

(21)

为了以 T4 上有一个路段以及所有其它的磁带都变空告终，我们需要有

$$v_0=1$$
$$u_0+v_1=0$$
$$u_1+v_2=u_0+v_0$$
$$u_2+v_3=u_1+v_1+u_0+v_0$$
$$u_3+v_4=u_2+v_2+u_1+v_1+u_0+v_0$$
$$u_4+v_5=u_3+v_3+u_2+v_2+u_1+v_1+u_0+v_0$$
$$u_5+v_6=u_4+v_4+u_3+v_3+u_2+v_2+u_1+v_1+u_0+v_0$$

等等；一般说来，要求对于所有的 $n\geqslant 0$，

$$u_n+v_{n+1}=u_{n-1}+v_{n-1}+u_{n-2}+v_{n-2}+u_{n-3}+v_{n-3}+u_{n-4}+v_{n-4} \quad (22)$$

这里我们认为，对所有 $j<0$ $u_j=v_j=0$。

这些方程没有惟一的解；其实，如果令所有的 u 都为 0，则我们就浪费了一条磁带而得到了通常的多阶段合并！但是如果我们选择 $u_n\approx v_{n+1}$，则重绕时间将令人满意地重叠。

McAllester 建议取

$$u_n=v_{n-1}+v_{n-2}+v_{n-3}+v_{n-4}$$
$$v_{n+1}=u_{n-1}+u_{n-2}+u_{n-3}+u_{n-4}$$

使得序列

$$\langle x_0, x_1, x_2, x_3, x_4, x_5, \cdots \rangle = \langle v_0, u_0, v_1, u_1, v_2, u_2, \cdots \rangle$$

满足一致递推式 $x_n = x_{n-3} + x_{n-5} + x_{n-7} + x_{n-9}$。然而可以证明,若令

$$\begin{aligned} v_{n+1} &= u_{n-1} + v_{n-1} + u_{n-2} + v_{n-2} \\ u_n &= u_{n-3} + v_{n-3} + u_{n-4} + v_{n-4} \end{aligned} \tag{23}$$

则更好,这个序列不仅在它的合并时间方面稍好些,而且它还有很大的长处,即对应的合并时间可从数学上加以分析。McAllester 的选择是极其难以分析的,因为不同长度的路段可以在同一个阶段中出现;我们将看到这对于(23)则不然。

在(21)的型式中按反方向进行,我们可以导出每级上每条磁带的路段数,并得到下列的排序方案:

级	T1	T2	T3	T4	T5	T6	写的时间	重绕时间
	1^{23}	1^{21}	1^{17}	1^{10}	—	1^{11}	82	23
7	1^{19}	1^{17}	1^{13}	1^{6}	R	$1^{11}4^{4}$	$4\times4=16$	82—23
	1^{13}	1^{11}	1^{7}	—	4^{6}	R	$6\times4=24$	27
6	1^{10}	1^{8}	1^{4}	R	4^{9}	$1^{8}4^{4}$	$3\times4=12$	10
	1^{6}	1^{4}	—	4^{4}	R	$1^{4}4^{4}$	$4\times4=16$	36
5	1^{5}	1^{3}	R	$4^{4}7^{1}$	4^{8}	$1^{3}4^{4}$	$1\times7=7$	17
	1^{2}	—	7^{3}	R	4^{5}	4^{4}	$3\times7=21$	23
4	1^{1}	R	$7^{3}13^{1}$	$4^{3}7^{1}$	4^{4}	4^{3}	$1\times13=13$	21
	—	13^{1}	R	$4^{2}7^{1}$	4^{3}	4^{2}	$1\times13=13$	34
3	R	$13^{1}19^{1}$	$7^{2}13^{1}$	$4^{1}7^{1}$	4^{2}	4^{1}	$1\times19=19$	23
	19^{1}	R	$7^{1}13^{1}$	7^{1}	4^{1}	—	$1\times19=19$	32
2	$19^{1}31^{0}$	$13^{1}19^{1}$	$7^{1}13^{1}$	7^{1}	4^{1}	R	$0\times31=0$	27
	R	19^{1}	13^{1}	7^{0}	—	31^{1}	$1\times31=31$	19
1	$19^{1}31^{0}$	19^{1}	13^{1}	7^{0}	R	$31^{1}52^{0}$	$0\times52=0$	
	$19^{1}31^{0}$	19^{1}	13^{1}	—	52^{0}	R	$0\times52=0$	max(36,31,23)
0	$19^{1}31^{0}$	19^{1}	13^{1}	R	$52^{0}82^{0}$	$31^{1}52^{0}$	$0\times82=0$	
	(31^{0})	(19^{0})	—	82^{1}	(R)	(52^{0})	$1\times82=82$	0

当输入磁带 T5 被重绕时(82 个单位),在第 2 级的上半阶段(27 个单位),以及在第 1 级和第 0 级的最后的"虚拟合并"阶段(36 个单位),在 3 处出现非重叠的重绕。所以我们可以估算这个时间是 $273t + 145r$;算法 D 相应的量是 $268t + 208r$,几乎总要低些。

习题 23 证明,在每个阶段,路段的输出长度逐次为

$$4, 4, 7, 13, 19, 31, 52, 82, 133, \cdots \tag{24}$$

这是一个满足规则

$$t_n = t_{n-2} + 2t_{n-3} + t_{n-4} \tag{25}$$

的序列，如果认为当 $n \leqslant 0$ 时 $t_n = 1$ 的话。通过像我们在等式(8)中对于标准的多阶段法所做的那样，来考察合并数的串，我们也可以分析虚拟路段的最优设置：

级	T1	T2	T3	T4	T6	最后输出于
1	1	1	1	1	—	T5
2	1	1	1	—	1	T4
3	21	21	2	2	1	T3
4	2221	222	222	22	2	T2
5	23222	23222	2322	23	222	T1
6	333323222	33332322	333323	3333	2322	T6
……						
n	A_n	B_n	C_n	D_n	E_n	T(k)
$n+1$	$(A''_nE_n+1)B_n$	$(A''_nE_n+1)C_n$	$(A''_nE_n+1)D_n$	A''_nE_n+1	A'_n	T($k-1$)
……						

(26)

其中 $A_n = A'_nA''_n$，且 A''_n 由 A_n 的最后 u_n 个合并数组成。上述从 n 级进行到 $n+1$ 级的规则对于满足(22)的任何方案都是正确的。当我们用(23)来定义 u 和 v 时，可以下列类似于(9)的颇简单的方式来表达串 $A_n, \cdots, E_n$：

$$\begin{aligned}
A_n &= (W_{n-1}W_{n-2}W_{n-3}W_{n-4}) + 1\\
B_n &= (W_{n-1}W_{n-2}W_{n-3}) + 1\\
C_n &= (W_{n-1}W_{n-2}) + 1\\
D_n &= (W_{n-1}) + 1\\
E_n &= (W_{n-2}W_{n-3}) + 1
\end{aligned} \tag{27}$$

其中

$$\begin{aligned}
W_n &= (W_{n-3}W_{n-4}W_{n-2}W_{n-3}) + 1 \quad \text{对于 } n > 0\\
W_0 &= 0 \text{ 和 } W_n = \epsilon \quad \text{对于 } n < 0
\end{aligned} \tag{28}$$

从这些关系，容易对 6 条磁带的情况进行详细的分析。

一般说来，当有 $T \geqslant 5$ 条磁带时，我们令 $P = T - 2$，而且通过规则

$$\begin{aligned}
v_{n+1} &= u_{n-1} + v_{n-1} + \cdots + u_{n-r} + v_{n-r}\\
u_n &= u_{n-r-1} + v_{n-r-1} + \cdots + v_{n-P} + v_{n-P} \quad \text{对于 } n \geqslant 0
\end{aligned} \tag{29}$$

定义序列 $\langle u_n \rangle$，$\langle v_n \rangle$，其中 $r = \lfloor P/2 \rfloor$；$v_0 = 1$ 且对 $n < 0$，$u_n = v_n = 0$。所以如果 $w_n = u_n + v_n$，则我们有

$$w_n = w_{n-2} + \cdots + w_{n-r} + 2w_{n-r-1} + w_{n-r-2} + \cdots + w_{n-P} \quad \text{对于 } n > 0 \tag{30}$$

$w_0=1$;且当 $n<0$ 时 $w_n=0$。$n+1$ 级的磁带上的初始分布是:置 $w_n+w_{n-1}+\cdots+w_{n-P+k}$个路段到磁带 k 上;$1\leqslant k\leqslant P$,以及置 $w_{n-1}+\cdots+w_{n-r}$个路段到磁带 T 上;磁带 $T-1$ 用于输入。然后当磁带 $T-1$ 正被重绕时,u_n 个路段被合并到 T 上,在 T 正重绕时,v_n 个路段被合并到 $T-1$ 上;当 $T-2$ 正重绕时,u_{n-1}个路段合并到 $T-1$ 上,等等。

当 S 不太小时,表 6 示出了这一过程的近似特性。"扫描/阶段"列近似地指出在一个阶段的每一半期间,整个文件中有多少正在被重绕,以及在每个完全的阶段,此文件近似地有多少正在被写出。*对于6条或更多的磁带,分带方法是比标准多阶段法要好的*,而且对于 5 条磁带或许也是这样,至少对于很大的 S 是这样。

表 6 使用分磁带的多阶段合并的近似行为

磁带	阶 段	扫 描	扫描/阶段	增长率
4	$2.855\ln S+0.000$	$1.443\ln S+1.000$	50%	1.4142136
5	$2.078\ln S+0.232$	$0.929\ln S+1.022$	45%	1.6180340
6	$2.078\ln S-0.170$	$0.752\ln S+1.024$	36%	1.6180340
7	$1.958\ln S-0.408$	$0.670\ln S+1.007$	34%	1.6663019
8	$2.008\ln S-0.762$	$0.624\ln S+0.994$	31%	1.6454116
9	$1.972\ln S-0.987$	$0.595\ln S+0.967$	30%	1.6604077
10	$2.013\ln S-1.300$	$0.580\ln S+0.941$	29%	1.6433803
20	$2.069\ln S-3.164$	$0.566\ln S+0.536$	27%	1.6214947

当 $T=4$ 时,上述过程实际上就等价于平衡的两路合并,没有重叠重绕的时间,因为对所有的 n,w_{2n+1}都将成为 0。所以略加修改,置 $v_2=0$,$u_1=0$,$v_1=0$,$u_0=0$,$v_0=1$,和对于 $n\geqslant 2$,$v_{n+1}=u_{n-1}+v_{n-1}$,$u_n=u_{n-2}+v_{n-2}$,即得到 $T=4$ 的表 6 的诸项。这导出了一个非常有趣的方案(见习题 25 和 26)。

习 题

1. [*16*] 图 69 所示为路段 34 到 65 通过算法 D 被分布到 5 条磁带上的次序;问路段 1 到 33 以什么次序分布?

▶2. [*21*] 真或假:在算法 D 中的两个合并阶段之后(即在我们第二次达到步骤 D6 时),所有的虚拟路段都已经消失。

▶3. [*22*] 证明在步骤 D4 的结果处,条件 $D[1]\geqslant D[2]\geqslant\cdots\geqslant D[T]$ 总是满足的。说明为什么在下述意义之下,这个条件是重要的,即:否则步骤 D2 和 D3 就不能有效工作。

4. [*M20*] 导出生成函数(7)。

5. [*HM26*] (E. P. Miles, Jr., 1960) 对于所有 $p\geqslant 2$,证明多项式 $f_p(z)=z^p-z^{p-1}-\cdots-z-1$ 有 p 个不同的根,它们当中恰有一个绝对值大于 1[提示:考虑多项式 $z^{p+1}-2z^p+1$]。

6. [*HM24*] 本题的目的是考虑如何来编制表 1、5 和 6。假定我们有一个合并型式,其性质以下列方式由多项式 $p(z)$和 $q(z)$表征:(i)出现在要求 n 个合并阶段的一个"完全分布"中的初始路段个数是$[z^n]p(z)/q(z)$。(ii)在这 n 个合并阶段里处理的初始路段个数是$[z^n]p(z)/q(z)^2$。(iii)有 $q(z^{-1})$的一个"最大"的根 α,使得 $q(\alpha^{-1})=0$, $q'(\alpha^{-1})\neq 0$, $p(\alpha^{-1})\neq 0$,而且 $q(\beta^{-1})=0$ 蕴涵着 $\beta=\alpha$ 或$|\beta|<|\alpha|$。

证明存在一个数 $\varepsilon>0$,使得如果 S 是在一个要求 n 个合并阶段的完全分布中的路段数,而且如果在这些阶段中处理了 ρS 个初始路段,则我们有 $n=a\ln S+b+O(S^{-\varepsilon})$, $\rho=c\ln S+d+O(S^{-\varepsilon})$,其中

$$a=(\ln\alpha)^{-1}, b=-a\ln\left(\frac{p(\alpha^{-1})}{-q'(\alpha^{-1})}\right)-1, c=a\frac{\alpha}{-q'(\alpha^{-1})}$$

$$d=\frac{(b+1)\alpha-p'(\alpha^{-1})/p(\alpha^{-1})+q''(\alpha^{-1})/q'(\alpha^{-1})}{-q'(\alpha^{-1})}$$

7. [*HM22*] 设 α_p 是习题 5 中多项式 $f_p(z)$的最大的根,问当 $p\to\infty$时,α_p 的渐近特性是什么?

8. [*M20*] (E. Netto, 1901) 设 $N_m^{(p)}$ 是把 m 表达成整数$\{1,2,\cdots,p\}$的一个有序和的方式数。例如,当 $p=3$ 和 $m=5$ 时,有 13 种方式,即 $1+1+1+1+1=1+1+1+2=1+1+2+1=1+1+3=1+2+1+1=1+2+2=1+3+1=2+1+1+1=2+1+2=2+2+1=2+3=3+1+1=3+2$。证明 $N_m^{(p)}$ 是一个广义斐波那契数。

9. [*M20*] 设 $K_m^{(p)}$ 是 m 个 0 和 1 的这样的序列的个数,即其中没有 p 个相邻的 1 出现。例如,当 $p=3$ 和 $m=5$ 时,有 24 种方式:00000,00001,00010,00011,00100,00101,00110,01000,01001,…,11011。证明 $K_m^{(p)}$ 是一个广义斐波那契数。

10. [*M27*] (广义斐波那契数系统)证明每一个非负整数 n 都可以惟一地表示为不同的 p 阶斐波那契数 $F_j^{(p)}$ 之和,$j\geqslant p$,而且这个表示不使用连续的 p 个斐波那契数。

11. [*M24*] 证明(12)中的串 Q_∞的第 n 个元素,等于用第 5 阶斐波那契数表示 $n-1$ 时其中不同斐波那契数的个数(参见习题 10)。

12. [*M18*] 试求矩阵

$$\begin{pmatrix}0&1&0&0&0\\0&0&1&0&0\\0&0&0&1&0\\0&0&0&0&1\\1&1&1&1&1\end{pmatrix}$$

的乘方与(1)中的完全斐波那契分布之间的联系。

▶13. [*22*] 证明完全斐波那契分布的下列相当奇特的性质:当最后的输出在 T 号磁带上时,则在每个其它磁带上的路段数都是奇数;当最后的输出在不同于 T 的某个磁带上时,则在该磁带上的路段数将是奇的,而在其它磁带上的将是偶的[参见(1)]。

14. [*M35*] 设 $T_n(x)=\sum_{k\geqslant 0}T_{nk}x^k$,其中 $T_n(x)$是在(16)中定义的多项式。

a)证明对于每个 k,都有一个数 $n(k)$,使得 $T_{1k}\leqslant T_{2k}\leqslant\cdots\leqslant T_{n(k)k}>T_{(n(k)+1),k}\geqslant\cdots$。

b)若 $T_{n'k'}<T_{nk'}$且 $n'<n$,证明对所有的 $k\geqslant k'$, $T_{n'k}\leqslant T_{nk}$。

c)证明有一个非减的序列$\langle M_n\rangle$,使得当$M_n\leqslant S<M_{n+1}$时$\sum_n(S)=\min_{j\geqslant1}\sum_j(S)$,但当$S\geqslant M_{n+1}$时$\sum_n(S)>\min_{j\geqslant1}\sum_j(S)$[参见(19)]。

15.[*M43*]证明或否定,$\sum_{n-1}(m)<\sum_n(m)$意味着$\sum_n(m)\leqslant\sum_{n+1}(m)\leqslant\sum_{n+2}(m)\leqslant\cdots$[这样一个结果将大大地简化表2的计算]。

16.[*M43*]确定具有最优虚拟路段分布的多阶段合并的渐近特性。

17.[*32*]证明或否定:通过把一个路段(在一条适当的磁带上)加到S个初始路段的分布上,来形成$S+1$个初始路段的分布,这个方法可用来散布一个最优多阶段分布的路段。

18.[*30*]如果我们坚持初始路段至多被放置在$T-1$条磁带上,则在处理的初始路段个数极小的意义下,最优多阶段分布是否产生最好的合并型式(忽略重绕时间)?

19.[*21*]试对6条磁带上的Caron多阶段排序,构造类似于(1)的一张表。

20.[*M24*]对6条磁带上的Caron多阶段排序,什么生成函数对应于(7)和(16)?类似于(9)和(27)的什么关系定义合并数的串?

21.[*11*]在(26)中的第7级上,将出现什么?

22.[*M21*]序列(24)的每一项近似地等于前面两项之和。这种现象对于序列中剩下的数是否成立?叙述并证明关于$t_n-t_{n-1}-t_{n-2}$的一个定理。

▶23.[*29*]如果把(23)变成$v_{n+1}=u_{n-1}+v_{n-1}+u_{n-2}$,$u_n=v_{n-2}+u_{n-3}+v_{n-3}+u_{n-4}+v_{n-4}$,则对(25),(27)和(28)应做什么改变?

24.[*HM41*]当把v_{n+1}定义为$u_{n-1}+v_{n-1}+\cdots+u_{n-P}+v_{n-P}$的前$q$项之和时,试对各种$P=T-2$及$0\leqslant q\leqslant 2P$计算分带的多阶段过程的渐近特性(正文中仅仅处理$q=2\lfloor P/2\rfloor$的情况;参见习题23)。

25.[*19*]说明:本小节末尾提到的在4条磁带上的分带多阶段合并将如何对32个初始路段排序(给出逐个阶段的分析,像正文中82个路段的6条磁带的例子那样)。

26.[*M21*]分析当$S=2^n$和当$S=2^n+2^{n-1}$时,在4条磁带上的分带多阶段合并的特性(见习题25)。

27.[*23*]在一个完全分布中一旦初始路段已经分布到一些磁带上,则多阶段策略只不过是"合并直到空":我们从所有非空输入磁带合并路段,直到它们之一变成空的为止;然后使用该磁带作为下一输出磁带,并且让以前的输出磁带作为输入之一。

是否不论初始路段如何分布,只要我们至少把它们分布到两条磁带上,则这个合并直到空的策略就总能完成(当然,一条磁带将保持为空,以便它能作为第一条输出磁带)?

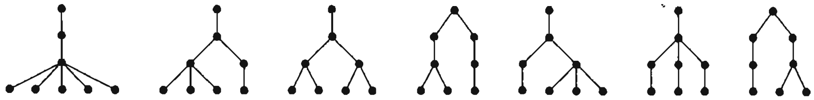
28.[*M26*]上边的习题定义了一族相当广泛的合并型式,说明在下列意义下多阶段法是其中最好的:如果有6条磁带,而且如果我们考虑所有这样的初始分布类(a,b,c,d,e),在这个类中合并直到空的策略要求最多n个阶段来进行排序,则$a+b+c+d+e\leqslant t_n$,其中t_n是对于多阶段排序(1)的相应的值。

29.[*M47*]习题28说明,在极少阶段的意义下,多阶段分布是在所有"合并直到成为空"的型式中最优的,但是在极少扫描的意义下它也是最优的吗?

设a和b互质,而且假定$a+b$是斐波那契数F_n。证明或否定下列的R.M.Karp的猜想:在以分布(a,b)开始的合并直到空型式期间处理的初始路段数,大于或等于$((n-5)F_{n+1}+(2n+2)F_n)/5$(当$a=F_{n-1}$,$b=F_{n-2}$时,达到此数)。

30.[*42*]对于分带多阶段合并,试编制类似表2的一张表。

31.[*M22*](R.Kemp),令$K_d(n)$是这样的n节点有序树的个数,即其中每个叶到根的距离为d。例如,$K_3(8)=7$,因为有下列的树:



证明 $K_d(n)$是一个广义的斐波那契数,并求这样的树和在习题 8 中所考虑的有序分划之间的一一对应。

*5.4.3 级联合并

另一个称为"级联合并"的基本型式,实际上在多阶段法之前就被发现了[B. K. Betz 和 W. C. Carter, *ACM National Conf.* **14**(1959), Paper 14]。沿用 5.4.2 小节所给出的记号,下表给出在 6 条磁带和 190 个初始路段的情况下使用该法的图示。

	T1	T2	T3	T4	T5	T6	处理的初始路段
扫描 1	1^{55}	1^{50}	1^{41}	1^{29}	1^{15}	—	190
扫描 2	—	$*1^{5}$	2^{9}	3^{12}	4^{14}	5^{15}	190
扫描 3	15^{5}	14^{4}	12^{3}	9^{2}	$*5^{1}$	—	190
扫描 4	—	$*15^{1}$	29^{1}	41^{1}	50^{1}	55^{1}	190
扫描 5	190^{1}	—	—	—	—	—	190

一个级联合并像多阶段法一样,以磁带上路段的一个"完全的分布"开始,尽管完全分布的规则有些不同于 5.4.2 小节中的那些。表中的每一行表示对所有数据的一次完全的扫描。例如,第二遍扫描是通过对{T1,T2,T3,T4,T5}进行一次五路合并到 T6,直到 T5 成为空为止(这把相对长度为 5 的 15 个路段放置到 T6 上),然后对{T1,T2,T3,T4}进行一次四路合并到 T5,然后进行三路合并到 T4,然后进行两路合并到 T3,最后从 T1 进行一路合并(一个拷贝操作)到 T2。第三遍扫描与此类似,首先进行一次五路合并直到 1 条磁带变空为止,然后进行一次四路合并,等等[也许,本书此节应编号为 5.4.3.2.1,而不是 5.4.3]。

显然拷贝操作是不必要的,故可以省略它们。然而,实际上,在 6 条磁带的情况下,这个拷贝仅仅花费总时间的很小的一部分。在上面的表中以星号标出的项目,就是简单地被拷贝的那些;处理的 950 个路段中仅有 25 个是这种类型的。大多数时间都花费在五路合并和四路合并上。

若与多阶段法进行比较,则第一个印象可能觉得级联型式是一种相当拙劣的选择,因为标准多阶段法始终使用 $T-1$ 路合并,而级联则使用 $T-1$ 路,$T-2$ 路,$T-3$ 路,等等。但事实上,在 6 条或更多的磁带上,它渐近地比多阶段法*更好*! 如同我们已经在 5.4.2 小节中发现的,一个高阶的合并并不保证高效性。通过同 5.4.2 小节中类似的表的类比,表 1 示出了级联合并的性能特征。

表 1 级联合并排序的近似特性

磁带	扫描(有拷贝)	扫描(没有拷贝)	增长率
3	$2.078 \ln S+0.672$	$1.504 \ln S+0.992$	1.6180340
4	$1.235 \ln S+0.754$	$1.102 \ln S+0.820$	2.2469796
5	$0.946 \ln S+0.796$	$0.897 \ln S+0.800$	2.8793852
6	$0.796 \ln S+0.821$	$0.773 \ln S+0.808$	3.5133371
7	$0.703 \ln S+0.839$	$0.691 \ln S+0.822$	4.1481149
8	$0.639 \ln S+0.852$	$0.632 \ln S+0.834$	4.7833861
9	$0.592 \ln S+0.861$	$0.587 \ln S+0.845$	5.4189757
10	$0.555 \ln S+0.869$	$0.552 \ln S+0.854$	6.0547828
20	$0.397 \ln S+0.905$	$0.397 \ln S+0.901$	12.4174426

通过由最后的状态(1,0,…,0)向后推演,容易导出对于一个级联合并的"完全分布"。对于 6 条磁带,它们是

级	T1	T2	T3	T4	T5
0	1	0	0	0	0
1	1	1	1	1	1
2	5	4	3	2	1
3	15	14	12	9	5
4	55	50	41	29	15
5	190	175	146	105	55
……	……	……	……	……	…… (1)
n	a_n	b_n	c_n	d_n	e_n
$n+1$	$a_n+b_n+c_n+d_n+e_n$	$a_n+b_n+c_n+d_n$	$a_n+b_n+c_n$	a_n+b_n	a_n

注意到这样一点是有趣的,即这些数的相对大小也出现在一个正 $2T-1$ 边多边形的对角线之中。例如,图 73 的 11 边形中的 5 条对角线有非常接近于 190,175,146,105 和 55 的相对长度!我们将在这一小节的稍后部分来证明这个值得注意的事实,而且我们也将看到,花费于$(T-1)$路合并,$(T-2)$路合并,…,1 路合并的相对时间近似地和这些对角线长度的平方成比例。

路段的初始分布 当初始路段的实际数目不完全时,我们可以像通常那样插入虚拟路段。对于这一情况的粗略分析指出,分派虚拟路段的方法是无所谓的,因为级联合并通过完整的扫描进行操作;如果我们有 190 个初始路段,则如同上面的例

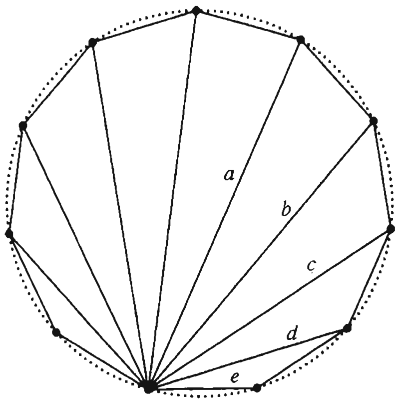


图 73 级联数的几何解释

子那样每个记录都被处理 5 次，但是如果有 191 个，则我们就必须上升一级使得每个记录都被处理 6 次。幸而这个急剧的改变并非真正必需；David E. Ferguson 已经找到了一种方法来分配初始路段，使得在第一趟合并期间的许多操作都归结为拷贝 1 条磁带的内容。当这样的拷贝关系被绕开（如像算法 5.4.2D 中那样，通过简单地改变同"物理"设备号相关的"逻辑"磁带设备号）时，我们就得到了从级到级的一个相当光滑的转换，如图 74 所示。

假设$(a,b,c,d,e,)$是一个完全的分布，其中 $a\geqslant b\geqslant c\geqslant d\geqslant e$。通过重新定义逻辑磁带和物理磁带设备之间的对应，我们可以想像这个分布实际上是(e,d,c,b,a)，有 a 个路段在 T5 上，有 b 个路段在 T4 上，等等。下一个完全的分布是$(a+b+c+d+e,a+b+c+d,a+b+c,a+b,a)$；而且如果在达到下一级之前已经穷尽了输入，则我们假定这些磁带分别包含(D_1,D_2,D_3,D_4,D_5)个虚拟路段，其中

$$D_1\leqslant a+b+c+d,\ D_2\leqslant a+b+c,\ D_3\leqslant a+b,\ D_4\leqslant a_1,\ D_5=0$$

$$D_1\geqslant D_2\geqslant D_3\geqslant D_4\geqslant D_5 \qquad (2)$$

我们现在可以自由地想像虚拟路段出现在这些磁带上的任何方便的位置。假定第一趟合并扫描通过五路合并产生 a 个路段，然后通过四路合并产生 b 个路段，等等。而我们的目标是安排这些虚拟路段以便用拷贝来代替合并。我们不妨按如下方式来做第一趟合并扫描：

1. 如果 $D_4=a$，则 D_1,D_2,D_3,D_4 中每一个都减去 a，并假定 T5 就是合并的结果。如果 $D_4<a$，从磁带 T1 到 T5 合并 a 个路段，并使用的这 5 条磁带上的极小个数的虚拟路段，使得 D_1,D_2,D_3,D_4 的新值满足

$$D_1\leqslant b+c+d,D_2\leqslant b+c,D_3\leqslant b,D_4=0;$$

$$D_1\geqslant D_2\geqslant D_3\geqslant D_4 \qquad (3)$$

例如，如果 D_2 原来$\leqslant b+c$，则在这一步我们不使用来自它的虚拟路段，而如果 $b+c<D_2\leqslant a+b+c$，则我们恰巧使用其中的 D_2-b-c 个。

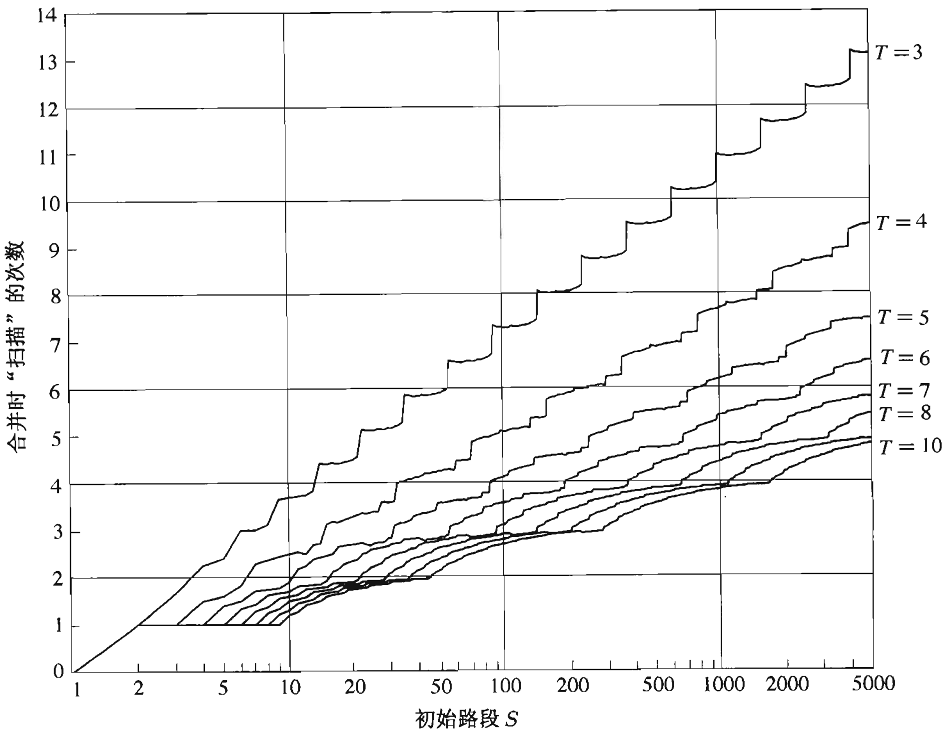


图 74 具有算法 D 的分布的级联合并的效率

2.(这一步类似于第一步,但已“移位”)如果 $D_3=b$,则 D_1,D_2,D_3 中的每一个都减去 b,而且设想 T4 就是合并的结果。如果 $D_3<b$,则从磁带 T1 到 T4 合并 b 个路段,必要时减少虚拟路段数以使得

$$D_1 \leqslant b+c, D_2 \leqslant b, D_3=0; \quad D_1 \geqslant D_2 \geqslant D_3$$

3.等等。

Ferguson 的分布路段到磁带的方法,可以通过考察(1)中从第 3 级进行到第 4 级的过程来说明。假设“逻辑”磁带(T1,…,T5)分别包含(5,9,12,14,15)个路段,而且我们最终要把它变成为(55,50,41,29,15)。这个过程可以概括为如表 2 所列。我们首先放置 9 个路段于 T1 上,然后置(3,12)到 T1 和 T2 上,等等。如果在比如说步骤(3,2)期间输入已经穷尽,则“节省的数量”是 15+9+5,意味着通过分配虚拟路段而避免了 15 个路段的五路合并,9 个路段的两路合并,以及 5 个路段的一路合并。换句话说,在第一合并阶段出现于第 3 级的路段中,有 15+9+5 个不予处理。

以下的算法详细地定义了这一进程。

表 2 级联分布步骤举例

	加到 T1	加到 T2	加到 T3	加到 T4	加到 T5	"节省的数量"
步(1,1)	9	0	0	0	0	15+14+12+5
步(2,2)	3	12	0	0	0	15+14+9+5
步(2,1)	9	0	0	0	0	15+14+5
步(3,3)	2	2	14	0	0	15+12+5
步(3,2)	3	12	0	0	0	15+9+5
步(3,1)	9	0	0	0	0	15+5
步(4,4)	1	1	1	15	0	14+5
步(4,3)	2	2	14	0	0	12+5
步(4,2)	3	12	0	0	0	9+5
步(4,1)	9	0	0	0	0	5

算法 C（具有特殊分布的级联合并排序） 这个算法取初始的路段并把它们分散到磁带上，一次一个路段，直到提供的初始路段被穷尽为止。然后它确定诸磁带如何被合并，假定有 $T\geqslant 3$ 台可供利用的磁带机，同时至多利用 $T-1$ 路合并，并避免不必要的一路合并。磁带 T 可用来保留输入，因为它不接受任何初始的路段。下列诸表是必需的：

A[j]，$1\leqslant j\leqslant T$：我们最近达到的完全级联分布。

AA[j]，$1\leqslant j\leqslant T$：我们正在争取的完全级联分布。

D[j]，$1\leqslant j\leqslant T$：假定在设备号为 j 的逻辑磁带上要出现的虚拟路段数。

M[j]，$1\leqslant j< T$：在设备号为 j 的逻辑磁带上希望的极大虚拟路段数。

TAPE[j]，$1\leqslant j\leqslant T$：对应于逻辑磁带设备号 j 的物理磁带设备号。

C1.［初始化］ 对于 $2\leqslant k\leqslant T$ 置 A[k]←AA[k]←D[k]←0；并置 A[1]←0，AA[1]←1，D[1]←1。对于 $1\leqslant k\leqslant T$ 置 TAPE[k]←k。最后，置 $i\leftarrow T-2, j\leftarrow 1, k\leftarrow 1, l\leftarrow 0, m\leftarrow 1$，并转到步骤 C5（这一处理手段是通过给控制变量赋适当的值，以便直接跳入内循环，从而一举启动整个过程的一种办法）。

C2.［开始新的一级］（我们刚才已经达到一个完全的分布，而且由于还有更多的输入，我们必须为下一级做好准备）l 增 1，对于 $1\leqslant k\leqslant T$ 置 A[k]←AA[k]，然后对于 $k=1,2,\cdots,T-1$，以这个顺序置 AA[$T-k$]←AA[$T-k+1$]+A[k]。置（TAPE[1]，…，TAPE[$T-1$]）←（TAPE[$T-1$]，…，TAPE[1]），并对于 $1\leqslant k<T$ 置 D[k]←AA[$k+1$]。最后置 $i\leftarrow 1$。

C3.［开始第 i 个子级］ 置 $j\leftarrow i$（变量 i 和 j 表示表 2 所示例子中的"步骤(i，j)"）。

C4.［开始步骤(i，j)］ 置 $k\leftarrow j$ 和 $m\leftarrow$A[$T-j-1$]。如果 $m=0$ 和 $i=j$，则置 $i\leftarrow T-2$ 并返回 C3；如果 $m=0$ 和 $i\neq j$，则返回 C2（变量 m 表示有待写到 TAPE[k]上的路段数；$m=0$ 仅当 $l=1$ 时才出现）。

C5.［输入到 TAPE[k]］ 写一个路段到号码为 TAPE[k]的磁带，且 D[k]减 1。然后如果输入已穷尽，则重绕所有的磁带并转到步骤 C7。

C6. [推进] m 减 1。如果 $m>0$,则返回 C5。否则 k 减 1;如果 $k>0$,则置 $m \leftarrow$ A[$T-j-1$] - A[$T-j$],并且若 $m>0$,则返回 C5。否则 j 减 1;如果 $j>0$,则转到 C4。否则 i 加 1;如果 $i<T-1$,则返回 C3。否则转回 C2。

C7. [准备合并] (这时初始分布已经完成,而且 AA、D 和 TAPE 诸表描述了磁带的现状) 对于 $1 \leqslant k < T$ 置 M[k]←AA[$k+1$],并置 FIRST←1(变量 FIRST 只在第一趟合并扫描期间非 0)。

C8. [级联] 如果 $l=0$,则停止;排序完成并输出在 TAPE[1]上。否则,以 $p=T-1, T-2, \cdots, 1$ 这个次序,进行从 TAPE[1],…,TAPE[p]到 TAPE[$p+1$]的 p 路合并如下:如果 $p=1$,则通过简单地重绕 TAPE[2],模拟一路合并,然后交换 TAPE[1]↔TAPE[2]。否则如果 FIRST = 1 且 D[$p-1$] = M[$p-1$],则通过简单地交换 TAPE[p]↔TAPE[$p+1$],重绕 TAPE[p],并从每个 D[1],…,D[$p-1$],M[1],…,M[$p-1$]都减去 M[$p-1$],模拟 p 路合并。否则从每个 M[1],…,M[$p-1$]减去 M[$p-1$]。然后从每个满足 $1 \leqslant j \leqslant p$ 且使 D[j]≤M[j]的 TAPE[j]合并一个路段;从每个满足 $1 \leqslant j \leqslant p$ 且使 D[j]>M[j]的 D[j]减 1;并且置输出路段到 TAPE[$p+1$]上。继续进行这一动作直到 TAPE(p)成为空为止。然后重绕 TAPE[p]和 TAPE[$p+1$]。

C9. [下降一级] l 减 1,置 FIRST←0,且置(TAPE[1],…,TAPE[T])←(TAPE[T],…,TAPE[1])(这时,所有的 D 和 M 皆为 0,而且将保持如此)。返回 C8。 ▌

此算法的 C1～C6 步进行分布,而 C7～C9 步进行合并;这两个部分彼此是完全独立的,而且将有可能在相同的存储单元中来保存 M[k]和 AA[$k+1$]。

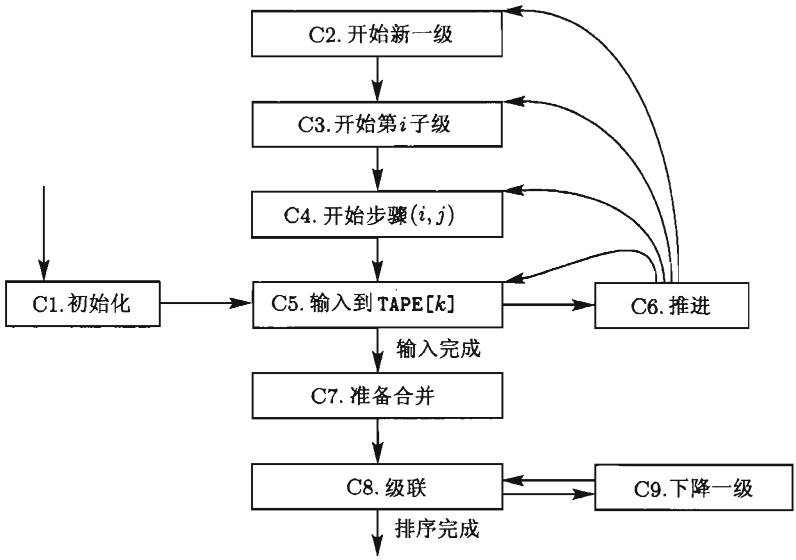


图 75 具有特殊分布的级联合并

级联合并的分析　比起多阶段的情况来,级联合并有些难以分析,但由于出现了许多引人注意的公式,因而这个分析是特别有趣的。建议喜欢离散数学的读者,在进一步阅读之前,自己先来研究级联分布,因为这些数具有那么多不寻常的性质,去发现它们是一件快事。在这里,我们将讨论众多分析方法之一,并且强调可以发现结果的方法。

为了方便,我们考虑 6 个磁带的情况,并找出可以推广到所有 T 的公式,关系(1)导出第一个基本的模式:

$$
\begin{aligned}
a_n &= a_n && = \binom{0}{0}a_n \\
b_n &= a_n - e_{n-1} \\
&= a_n - a_{n-2} && = \binom{1}{0}a_n - \binom{2}{2}a_{n-2} \\
c_n &= b_n - d_{n-1} \\
&= b_n - a_{n-2} - b_{n-2} && = \binom{2}{0}a_n - \binom{3}{2}a_{n-2} + \binom{4}{4}a_{n-4} \\
d_n &= c_n - c_{n-1} \\
&= c_n - a_{n-2} - b_{n-2} - c_{n-2} && = \binom{3}{0}a_n - \binom{4}{2}a_{n-2} + \binom{5}{4}a_{n-4} - \binom{6}{6}a_{n-6} \\
e_n &= d_n - b_{n-1} \\
&= d_n - a_{n-2} - b_{n-2} - c_{n-2} - d_{n-2} && = \binom{4}{0}a_n - \binom{5}{2}a_{n-2} + \binom{6}{4}a_{n-4} - \binom{7}{6}a_{n-6} + \binom{8}{8}a_{n-8}
\end{aligned}
\tag{4}
$$

设 $A(z) = \sum_{n\geqslant 0} a_n z^n, \cdots, E(z) = \sum_{n\geqslant 0} e_n z^n$,并定义多项式

$$
\begin{aligned}
q_m(z) &= \binom{m}{0} - \binom{m+1}{2}z^2 + \binom{m+2}{4}z^4 - \cdots = \\
&\sum_k \binom{m+k}{2k}(-1)^k z^{2k} = \sum_{k=0}^{m}\binom{2m-k}{k}(-1)^{m-k}z^{2m-2k}
\end{aligned}
\tag{5}
$$

(4)的结果可概括如下:生成函数 $B(z)-q_1(z)A(z)$, $C(z)-q_2(z)A(z)$, $D(z)-q_3(z)A(z)$和 $E(z)-q_4(z)A(z)$简化为对应于 $a_{-1}, a_{-2}, a_{-3}\cdots$的值的有限和,对于小的 n,它们出现在(4)中但不出现在 $A(z)$中。为了提供适当的边界条件,让我们向后往负的方向向后运行递归式直到 −8 级:

n	a_n	b_n	c_n	d_n	e_n
0	1	0	0	0	0
−1	0	0	0	0	1
−2	1	−1	0	0	0
−3	0	0	0	−1	2
−4	2	−3	1	0	0
−5	0	0	1	−4	5

−6	5	−9	5	−1	0
−7	0	−1	6	−14	14
−8	14	−28	20	−7	1

在 7 个磁带上，这张表也是类似的，只是奇数 n 的项右移一列。序列 $a_0, a_{-2}, a_{-4}, \cdots = 1,1,2,5,14,\cdots$ 对于计算机科学家来说是司空见惯的，因为它同如此多的递归算法相关联而出现(例如习题 2.2.1-4 和等式 2.3.4.4-(14))；我们因此猜想在 T-磁带的情况下

$$\begin{aligned} a_{-2n} &= \binom{2n}{n}\frac{1}{n+1} \qquad \text{对于 } 0 \leqslant n \leqslant T-2 \\ a_{-2n-1} &= 0 \qquad\qquad\quad \text{对于 } 0 \leqslant n \leqslant T-3 \end{aligned} \tag{6}$$

为了验证这个选择是正确的，只要证明(6)和(4)对于第 0 级和第 1 级产生正确的结果即可。在第 1 级上，这是显然的，而在第 0 级上，我们要对 $0 \leqslant m \leqslant T-2$，验证

$$\binom{m}{0}a_0 - \binom{m+1}{2}a_{-2} + \binom{m+2}{4}a_{-4} - \binom{m+3}{6}a_{-6} + \cdots = \sum_{k\geqslant 0}\binom{m+k}{2k}\binom{2k}{k}\frac{(-1)^k}{k+1} = \delta_{m0} \tag{7}$$

幸而这个和可以通过标准的技术来求值；事实上，它是 1.2.6 小节的例 2。

现在我们可以计算 $B(z) - q_1(z)A(z)$ 等的系数。例如，考虑 $D(z) - q_3(z)A(z)$ 中 z^{2m} 的系数：由 1.2.6 小节中例 3 的结果，它是

$$\sum_{k\geqslant 0}\binom{3+m+k}{2m+2k}(-1)^{m+k}a_{-2k} = \sum_{k\geqslant 0}\binom{3+m+k}{2m+2k}\binom{2k}{k}\frac{(-1)^{m+k}}{k+1} = (-1)^m\left[\binom{2+m}{2m-1} - \binom{3+m}{2m}\right] = (-1)^{m+1}\binom{2+m}{2m}$$

因此我们已经推出

$$\begin{aligned} A(z) &= q_0(z)A(z) \\ B(z) &= q_1(z)A(z) - q_0(z) \qquad C(z) = q_2(z)A(z) - q_1(z) \\ D(z) &= q_3(z)A(z) - q_2(z) \qquad E(z) = q_4(z)A(z) - q_3(z) \end{aligned} \tag{8}$$

进而我们有 $e_{n+1} = a_n$；因此 $zA(z) = E(z)$，而且

$$A(z) = q_3(z)/(q_4(z) - z) \tag{9}$$

借助于 q 多项式，现在已经推导出了生成函数，因而我们想要对 q 有更好的了解。习题 1.2.9-15 在这方面是有用的，因为它给了我们一个“封闭的”形式，它可以写成

$$q_m(z) = \frac{((\sqrt{4-z^2} + \mathrm{i}z)/2)^{2m+1} + ((\sqrt{4-z^2} - \mathrm{i}z)/2)^{2m+1}}{\sqrt{4-z^2}} \tag{10}$$

如果我们现在置 $z=2\sin\theta$ 则一切都简化了：

$$q_m(2\sin\theta)=\frac{(\cos\theta+\mathrm{i}\sin\theta)^{2m+1}+(\cos\theta-\mathrm{i}\sin\theta)^{2m+1}}{2\cos\theta}=\frac{\cos(2m+1)\theta}{\cos\theta} \tag{11}$$

这种巧合导致我们猜想多项式 $q_m(z)$ 在数学上是已知的；确实，看一下适当的表将发现，$q_m(z)$ 实质上是第二类的契比雪夫多项式，即是通常符号下的 $(-1)^m U_{2m}(z/2)$。

我们现在可以确定(9)中分母的根：$q_4(2\sin\theta)=2\sin\theta$ 归结为

$$\cos 9\theta=2\sin\theta\cos\theta=\sin 2\theta$$

当 $\pm 9\theta=2\theta+\left(2n-\frac{1}{2}\right)\pi$ 时，我们能得到这个关系式的解；而且若 $\cos\theta=0$，则所有这样的 θ 都提供(9)中分母的根(当 $\cos\theta=0$ 时，$q_m(\pm 2)=\pm(2m+1)$ 决不等于 ± 2)。因此得到 $q_4(z)-z=0$ 的下列 8 个不同的根：

$$2\sin\frac{-5}{14}\pi,\ 2\sin\frac{-1}{14}\pi,\ 2\sin\frac{3}{14}\pi;\ 2\sin\frac{-7}{22}\pi,$$
$$2\sin\frac{-3}{22}\pi,\ 2\sin\frac{1}{22}\pi,\ 2\sin\frac{5}{22}\pi,\ 2\sin\frac{9}{22}\pi$$

由于 $q_4(z)$ 是 8 次多项式，这已枚举了所有的根。这些值的前 3 个值使得 $q_3(z)=0$，所以 $q_3(z)$ 和 $q_4(z)-z$ 有一个三次多项式作为公因子。如果我们展开(9)为部分分式，则其余 5 个根支配了 $A(z)$ 的系数的渐近特性。

在考虑一般的 T-磁带的情况下，令 $\theta_k=(4k+1)\pi/(4T-2)$。$T$-磁带级联分布数的生成函数 $A(z)$ 取形式(见习题 8)

$$\frac{4}{2T-1}\sum_{-T/2<k<\lfloor T/2\rfloor}\frac{\cos^2\theta_k}{1-z/(2\sin\theta_k)} \tag{12}$$

因此

$$a_n=\frac{4}{2T-1}\sum_{-T/2<k<\lfloor T/2\rfloor}\cos^2\theta_k\left(\frac{1}{2\sin\theta_k}\right)^n \tag{13}$$

等式(8)现在导致类似的公式

$$b_n=\frac{4}{2T-1}\sum_{-T/2<k<\lfloor T/2\rfloor}\cos\theta_k\cos 3\theta_k\left(\frac{1}{2\sin\theta_k}\right)^n$$
$$c_n=\frac{4}{2T-1}\sum_{-T/2<k<\lfloor T/2\rfloor}\cos\theta_k\cos 5\theta_k\left(\frac{1}{2\sin\theta_k}\right)^n \tag{14}$$
$$d_n=\frac{4}{2T-1}\sum_{-T/2<k<\lfloor T/2\rfloor}\cos\theta_k\cos 7\theta_k\left(\frac{1}{2\sin\theta_k}\right)^n$$

等等。习题 9 说明对于所有 $n\geqslant 0$ 而不仅仅对于大的 n，这些等式都成立。在每个和式中，$k=0$ 的项高于所有其余的项，特别是当 n 相当大时；因此“增长率”为

$$\frac{1}{2\sin\theta_0}=\frac{2}{\pi}T-\frac{1}{\pi}+\frac{\pi}{48T}+O(T^{-2}) \tag{15}$$

W.C.Carter 首先分析了级联排序[*Proc. IFIP Congress* (1962),62～66],他得到了对于小的 T 的数值结果,David E. Ferguson 也进行了分析[见*CACM* **7**(1964),297],他发现了增长率的渐近特性(15)中的前两项。1964 年夏,R.W.Floyd 发现了增长率的明显的形式 $1/(2\sin\theta_0)$,于是对于所有的 T 都可以使用准确的公式。G.N.Raney独立进行了级联数的一个深入细致的分析[*Canadian J. Math*. **18**(1966),332～349],他以与排序毫无关系的完全不同的方式来研究它们。Raney 观察了图 73 的"对角线比率"原理,而且导出了这些数的许多其它有趣性质。Floyd 和 Raney 在他们的证明中使用了矩阵处理(见习题 6)。

级联排序的修改 如果再加上一条磁带,则在级联排序期间,就有可能覆盖几乎所有的重绕时间。例如,我们可以把 T1～T5 合并到 T7,然后合并 T1～T4 到 T6,然后合并 T1～T3 到 T5(它现在已被重绕),然后合并 T1～T2 到 T4,而当已经重绕了 T4 上的比较短的数据时,即可开始下趟扫描。从对级联的上述分析中,即可预测这一过程的效率(关于进一步的信息见 5.4.6 小节)。

在*CACM* **6**(1963),585～587 中,D.E.Knuth 提出了一个"折中合并"的方案,它包括多阶段和级联两者作为特殊情况。每个阶段都由 $T-1$ 路,$T-2$ 路,…,P 路合并组成,其中 P 是 1 和 $T-1$ 之间的任何固定数。当 $P=T-1$ 时,这是多阶段的。而当 $P=1$ 时,它是纯粹的级联。当 $P=2$ 时,它是没有拷贝阶段的级联。C.E.Radke已经对这个方案进行了分析[*IBM Systems J*. **5**(1966),226～247]。W.H.Burge也进行了分析[*Proc IFIP Congress* (1971),**1**,454～459]。Burge 找到了对于每个(P,T)折中合并的生成函数$\sum T_n(x)z^n$,并推广了等式 5.4.2-(16);他证明,当 $S\to\infty$时,从作为 S 的一个函数来处理的最少初始路段的观点(利用一个直截了当的分布方案,并忽略重绕时间),则对于 $T=(3,4,5,6,7,8,9,10,11)$,P 的最好值分别为(2,3,3,4,4,4,3,3,4)。当 T 增加时P 的这些值更接近于级联而不是更接近于多阶段;由此得出,折中合并决不实质性地好于级联。另一方面,如同 5.4.2 小节所述,在选择最优的级数和最优的虚拟路段分布的情况下,纯粹的多阶段法似乎在所有折中合并中是最好的。不幸的是,最优分布比较难以实现。

Th.L.Johnsen[*BIT* **6**(1966),129～143]研究了平衡的和多阶段合并的组合;M.A.Goetz 提出平衡合并的一个重绕重叠的变种[*Digital Computer User's Handbook*, M.Klerer 和 G.A.Korn 编(NewYork:McGraw-Hill,1967),1.311～1.312];另外,还可以想像出许多其它混合的方案。

习 题

1. [*10*] 利用表 1,试比较级联合并和 5.4.2 小节中所述的多阶段法的分带方案。问哪一个更好(忽略重绕时间)?

▶2. [*22*] 比较在 3 条磁带上利用算法 C 的级联排序和利用算法 5.4.2D 的多阶段的排序,它

们之间有什么类似性和差别？

3. [*23*] 编制一张表，说明当在 6 条磁带上利用算法 C 对 100 个初始路段排序时所发生的情况。

4. [*M20*] (G. N. Raney)　一个“第 n 级级联分布”是如下定义的多重集合(在 6 条磁带的情况下)：$\{1,0,0,0,0\}$ 是第 0 级级联分布；而且如果 $\{a,b,c,d,e\}$ 是第 n 级级联分布，则 $\{a+b+c+d+e, a+b+c+d, a+b+c, a+b, a\}$ 是第 $n+1$ 级级联分布(因为一个多重集合是未排序的，从单单一个第 n 级的分布可以形成达 5! 种不同的第 $(n+1)$ 级分布)。(a)证明，互质整数的任何多重集合 $\{a,b,c,d,e\}$，都是对于某个 n 的第 n 级级联分布。(b)证明，在下列意义下，对于级联排序定义的分布是最优的，即如果 $\{a,b,c,d,e\}$ 是使 $a\geq b\geq c\geq d\geq e$ 的任何第 n 级分布，则我们有 $a\leq a_n, b\leq b_n, c\leq c_n, d\leq d_n, e\leq e_n$，其中 $\{a_n,b_n,c_n,d_n,e_n\}$ 是在(1)中定义的分布。

▶5. [*20*] 证明(1)中定义的级联数满足定律

$$a_ka_{n-k}+b_kb_{n-k}+c_kc_{n-k}+d_kd_{n-k}+e_ke_{n-k}=a_n\quad 对于\ 0\leq k\leq n$$

提示：通过考虑在一个完整的级联排序的第 k 趟扫描期间，共输出了多少各种长度的路段，来解释这个关系式。

6. [*M20*] 求一个 5×5 矩阵 Q，使得对于所有 $n\geq 0$，Q^n 的第一行包含 6 条磁带的级联数 a_n, b_n, c_n, d_n, e_n。

7. [*M20*] 假定级联合并被应用于 a_n 个初始路段的一个完全分布，试求当禁止一路合并时所节省下的工作量的公式。

8. [*HM23*] 推导(12)。

9. [*HM26*] 推导(14)。

▶10. [*HM28*] 不使用型式(4)来开始级联数的研究，而由恒等式

$$e_n=a_{n-1}\qquad\qquad\qquad =\binom{1}{1}a_{n-1}$$

$$d_n=2a_{n-1}-e_{n-2}\qquad\quad =\binom{2}{1}a_{n-1}-\binom{3}{3}a_{n-3}$$

$$c_n=3a_{n-1}-d_{n-2}-2e_{n-2}=\binom{3}{1}a_{n-1}-\binom{4}{3}a_{n-3}-\binom{5}{5}a_{n-5}$$

等开始，令

$$r_m(z)=\binom{m}{1}z-\binom{m+1}{3}z^3+\binom{m+2}{5}z^5-\cdots$$

用这些 r 多项式来表达 $A(z), B(z)$，等等。

11. [*M38*] 设

$$f_m(z)=\sum_{k=0}^{m}\binom{\lfloor (m+k)/2\rfloor}{k}(-1)^{\lceil k/2\rceil}z^k$$

证明，T-磁带级联数的生成函数 $A(z)$ 等于 $f_{T-3}(z)/f_{T-1}(z)$，其中这个表达式中的分子和分母没有公因子。

12. [*M40*] 证明在下面的意义下，Ferguson 的分布方案是最优的。即在第一次扫描期间任何满足(2)的分配虚拟路段的方法，都不能减少需要处理的初始路段个数，假定在这个扫描期间使用步骤 C7～C9 的策略。

13. [*40*] 正文中提议通过增加一条额外的磁带来覆盖大多数重绕时间。试剖析这个思想(例如，正文中的方案包含等候 T4 重绕；如果从下次扫描的第一个合并阶段省略 T4，是否将更好些)。

*5.4.4 向后读带

许多磁带机都有从同写入的方向相反的方向来读带的能力。我们迄今遇到的合并型总是以“向前”的方向把信息写到磁带上，然后重绕磁带，向前读，然后再次重绕它(因此磁带文件像队列那样，以先进先出的方式来操作)。而向后读允许我们消除上面的两个重绕操作：即我们向前写磁带和向后读它。在这种情况下，文件像栈那样处理，因为它们以后进先出的方式被使用。

平衡的、多阶段的，以及级联合并型式都能够适应向后读。主要差别是，当我们向后读和向前写时，*合并颠倒了路段的顺序*。如果两个路段在磁带上有递增的次序，则我们在向后读时可以合并它们，但这产生了递降的次序。这样产生的递降的次序，在下次扫描时将随继变成为递增的次序，所以合并算法必须有能力来处理递增或递降次序的路段。一个第一次向后读的程序员会经常感到他自己是倒立着的！

作为向后读的一个例子，考虑利用平衡合并在 4 条磁带上合并 8 个初始路段。这些操作可以综述如下：

	T1	T2	T3	T4	
扫描 1	$A_1A_1A_1A_1$	$A_1A_1A_1A_1$	—	—	初始分布
扫描 2	—	—	D_2D_2	D_2D_2	合并到 T3 和 T4
扫描 3	A_4	A_4	—	—	合并到 T1 和 T2
扫描 4	—	—	D_8	—	最后合并到 T3

这里 A_r 代表相对长度为 r 的路段，如果像我们以前的例子那样向前读这条磁带，则它的次序是递增的；D_r 代表长度为 r 的递降的相应路段。在第 2 次扫描期间递增的路段变成递降的；在输入时它们表现为递降的，因为我们在向后读。在第 3 次扫描时，它们又转换了方向。

注意，上边的过程以按*递降*次序出现在磁带 T3 上而结束。如果这样不行(依赖于输出是否向后读，还是被卸下并取走以供将来使用)，则我们可以把它拷贝到另一条磁带上，同时逆转方向。一种较快的方式是在第 3 次扫描后重绕 T1 和 T2，并且在第 4 次扫描时产生 A_8。还有一种更快的方式是在第一次扫描期间从 8 个递降路段开始，因为这将交换所有的 A 和 D。然而，对于 16 个初始路段的平衡的合并将要求初始路段是递增的；而且由于我们通常预先并不知道将形成多少初始路段，因此有必要来选择一个一致的方向。所以，在第 3 次扫描之后重绕的思想，大概是最好的。

级联合并以相同的方式进行。例如，考虑在 4 条磁带上对 14 个初始路段排序：

	T1	T2	T3	T4
扫描 1	$A_1A_1A_1A_1A_1A_1$	$A_1A_1A_1A_1A_1$	$A_1A_1A_1$	—
扫描 2	—	D_1	D_2D_2	$D_3D_3D_3$
扫描 3	A_6	A_5	A_3	—
扫描 4	—	—	—	D_{14}

又是如此:如果我们恰在最后一次扫描之前重绕 T1,T2,T3,则我们可以产生 A_{14}而不是 D_{14}。注意,在所有的一路合并都已明显执行的意义下,这是一个“纯粹的”级联合并,如果刚才像算法 5.4.3C 那样禁止拷贝操作,则在第 2 次扫描后我们将面临情况

$$A_1 \qquad — \qquad D_2D_2 \qquad D_3D_3D_3$$

而且已不可能继续三路合并,因为我们不可能合并处于相反方向的路段!如果我们重绕 T1 而且在下一个合并期间向前读它(同时向后读 T3 和 T4),则可避免拷贝 T1 到 T2 的操作。但在合并之后再次需要重绕 T1,所以这技巧以两次重绕来交换一次拷贝。

因此算法 5.4.3.C 的分布方法对于向后读并不像向前读那样有效;每当初始路段数通过一个“完全”的级联分布数时,所需要的时间数量有一颇为急剧的跳跃。另一种散布的技术可用来给出在完全的级联分布之间的一个更为光滑的转换(见习题 17)。

向后读的多阶段法 乍一看去(甚至第二次和第三次看去),多阶段合并方案似乎完全不适合于向后读。例如,假设有 13 个初始路段和 3 条磁带:

	T1	T2	T3
阶段 1	$A_1A_1A_1A_1A_1$	$A_1A_1A_1A_1A_1A_1A_1A_1$	—
阶段 2		$A_1A_1A_1$	$D_2D_2D_2D_2D_2$

现在我们停住;我们可以重绕 T2 或 T3,然后向前读它,同时向后读其它磁带,但这将得到颇为混杂的情况,而且通过向后读我们获益颇小。

一种弥补这种情况的巧妙想法是交替每条磁带上的路段方向。这样,合并即可完全同步地进行:

	T_1	T_2	T_3
阶段 1	$A_1D_1A_1D_1A_1$	$D_1A_1D_1A_1D_1A_1D_1A_1$	—
阶段 2	—	$D_1A_1D_1$	$D_2A_2D_2A_2D_2$
阶段 3	$A_3D_3A_3$	—	D_2A_2
阶段 4	A_3	D_5A_5	—
阶段 5	—	D_5	D_8
阶段 6	A_{13}	—	—

这一原理是由 R.L.Gilstad 在他关于多阶段合并的开创性论文中简略地提到的,他在*CACM* **6**(1963),220~223 中又做了更详细的描述。

对于任何数目的磁带上的多阶段合并,*ADA*…技术都能有效地工作;因为我们可以证明,在每个阶段 A 和 D 都将被适当地同步,为此仅仅假定初始的分布扫描在每条磁带上产生交替的 A 和 D,而且每条磁带都以 A 结束(或者每条磁带以 D 结束):因为在一个阶段期间写到输出文件上的最后路段,和由输入文件所取来的最后路段的方向相反,因此下一阶段总发现它的路段处于适当的方向。进而我们在习题 5.4.2-13 中已经看到,大多数完全的斐波那契分布都要求在一条磁带上有奇数个路

段(最后的输出磁带),而在其它的磁带上都有偶数个路段。如果把 T1 指定成最后的输出磁带,则我们因此能够保证,如果 T1 以 A 开始,而且剩下的磁带都以 D 开始,则所有的磁带都以 A 路段告终。可以使用类似于算法 5.4.2D 的一个分布方法,但修改成使在每级上的分布以 T1 作为最后的输出磁带(我们跳过级 1, $T+1$, $2T+1$, …,因为在这些级中,起始为空的磁带是最后的输出磁带)。例如,在 6 条磁带的情况下,我们可以使用下列的分布数代替 5.4.2-(1):

级	T1	T2	T3	T4	T5	总共	最后的输出将在
0	1	0	0	0	0	1	T1
2	1	2	2	2	2	9	T1
3	3	4	4	4	2	17	T1
4	7	8	8	6	4	33	T1
5	15	16	14	12	8	65	T1
6	31	30	28	24	16	129	T1
8	61	120	116	108	92	497	T1

(1)

于是,T1 总得到奇数个路段,而 T2 到 T5 得到偶数个路段,路段按递降次序排列,这是为了在支配虚拟路段时具有灵活性。这样一种分布有其优点,即最后的输出磁带是预先知道的,而不管有待出现的初始路段数是多少。结果(见习题 3),当使用这个方案时,输出将总是以递增的次序出现在 T1 上。

D.T. Goodwin 和 J.L. Venn 已经提出了用于处理向后读多阶段分布的另一个方法[*CACM* **7**(1964),315]。若每条磁带上都以一个 D 路段开始,我们几乎可以像在算法 5.4.2D 中那样分布路段。当耗尽输入时,除非已经达到对全部奇数编号的磁带的分布了,否则即想像有一个虚拟 A 路段在惟一的"奇数"磁带开始处。其它的虚拟路段想像成都在磁带的末端,或者组成为位于中间的对偶。下面的习题 5 中分析了虚拟路段的最优放置问题。

最优合并型式 迄今我们已经讨论了关于合并到磁带上的各种型式,但没有问及"最好"的方法。要确定最优的型式似乎是十分困难的,特别是在向前读的情况下,其中重绕时间同合并时间的相互作用难以处理。另一方面,当通过向后读和向前写完成合并时,所有重绕实质上都被消去,而且有可能得到相当好的最优合并型式的特性。Richard M. Karp 引进了某些非常有趣的方法来解决这个问题,我们将通过讨论他所提出的理论来结束这一小节。

首先,我们需要一种更令人满意的描述合并型式的方式,来替代前面所使用的"磁带内容"图。Karp 提出了两种方法来做这件事,即合并型式的*向量表示*和*树表示*。这两种表示形式实际上都是有用的,所以我们将依次来描述它们。

一个合并型式的*向量表示*,由一系列"合并向量" $y^{(m)} \cdots y^{(1)} y^{(0)}$ 所组成,它们每一个都有 T 个分量。以如下方式用 $y^{(i)}$ 来表示倒数第 i 个合并步:

$$y_j^{(i)} = \begin{cases} +1 & \text{如果第 } j \text{ 号磁带是对于合并的一个输入} \\ 0 & \text{如果第 } j \text{ 号磁带在合并中未用} \\ -1 & \text{如果第 } j \text{ 号磁带得到合并的输出} \end{cases} \qquad (2)$$

于是，$y^{(i)}$恰有一个分量为 -1，而其余的分量为 0 和 1。最后的向量 $y^{(0)}$是特殊的；它是单位向量，如果最后的排序输出出现在第 j 号设备处，则该向量在位置 j 处有 1，其余处则为 0。此定义意味着，向量和

$$v^{(i)} = y^{(i)} + y^{(i-1)} + \cdots + y^{(0)} \tag{3}$$

表示倒数第 i 个合并步之前磁带上路段的分布。特别是，$v^{(m)}$说明初始分布扫描安置多少个路段到每条磁带上。

把这些向量倒着编号，即从 $y^{(m)}$开始而以 $y^{(0)}$结尾，似乎并不方便，但是这种特殊的观点对发展这一理论是有利的。为了寻找一个最优的方法，一个好办法就是以排好序的输出开始，并且想像把它拆开到各条磁带上，然后再拆开这些，等等，同时，以同它们在排序过程中实际出现的顺序相颠倒的顺序，考虑依次的分布 $v^{(0)}$，$v^{(1)}$，$v^{(2)}$，…。实质上，这就是我们在对多阶段和级联合并进行分析时，已经采取的方法。

在本节中最先以表格形式描述的 3 种合并型式有下列向量表示：

平衡的（$T=4,S=8$）	级联（$T=4,S=14$）	多阶段（$T=3,S=13$）
$v^{(7)}=(\ 4,\ 4,\ 0,\ 0)$	$v^{(10)}=(\ 6,\ 5,\ 3,\ 0)$	$v^{(12)}=(\ 5,\ 8,\ 0)$
$y^{(7)}=(+1,+1,-1,\ 0)$	$y^{(10)}=(+1,+1,+1,-1)$	$y^{(12)}=(+1,+1,-1)$
$y^{(6)}=(+1,+1,\ 0,-1)$	$y^{(9)}=(+1,+1,+1,-1)$	$y^{(11)}=(+1,+1,-1)$
$y^{(5)}=(+1,+1,-1,\ 0)$	$y^{(8)}=(+1,+1,+1,-1)$	$y^{(10)}=(+1,+1,-1)$
$y^{(4)}=(+1,+1,\ 0,-1)$	$y^{(7)}=(+1,+1,-1,\ 0)$	$y^{(9)}=(+1,+1,-1)$
$y^{(3)}=(-1,\ 0,+1,+1)$	$y^{(6)}=(+1,+1,-1,\ 0)$	$y^{(8)}=(+1,+1,-1)$
$y^{(2)}=(\ 0,-1,+1,+1)$	$y^{(5)}=(+1,-1,\ 0,\ 0)$	$y^{(7)}=(-1,+1,+1)$
$y^{(1)}=(+1,+1,-1,\ 0)$	$y^{(4)}=(-1,+1,+1,+1)$	$y^{(6)}=(-1,+1,+1)$
$y^{(0)}=(\ 0,\ 0,\ 1,\ 0)$	$y^{(3)}=(\ 0,-1,+1,+1)$	$y^{(5)}=(-1,+1,+1)$
	$y^{(2)}=(\ 0,\ 0,-1,+1)$	$y^{(4)}=(+1,-1,+1)$
	$y^{(1)}=(+1,+1,+1,-1)$	$y^{(3)}=(+1,-1,+1)$
	$y^{(0)}=(\ 0,\ 0,\ 0,\ 1)$	$y^{(2)}=(+1,+1,-1)$
		$y^{(1)}=(-1,+1,+1)$
		$y^{(0)}=(\ 1,\ 0,\ 0)$

每个合并型式显然有一个向量表示。反之，容易看到，向量序列 $y^{(m)}\cdots y^{(1)}y^{(0)}$对应于一个实在的合并型式的充要条件，为下列 3 点：

i）$y^{(0)}$是一个单位向量。

ii）对于 $m\geqslant i\geqslant 1$，$y^{(i)}$恰有一个分量等于 -1，所有其余的分量为 0 或 $+1$。

iii）对于 $m\geqslant i\geqslant 1$，$y^{(i)}+\cdots+y^{(1)}+y^{(0)}$的所有分量非负。

一个合并型式的“树表示”给出了同样信息的另一种图示。我们构造一株树，对于每个初始路段有一个外部“叶”节点，对于被合并的每个路段有一个内部节点，构造的方式是：每个内部节点的后裔是那样一些路段，该内部节点是从这些路段制作出来的。每个内部节点都以对应的开始时的步骤号作为标号，并且像在向量表示中

那样，步骤是向后编号的；进而，每个节点上面的边以路段所在的磁带名来标出。例如，上述的3种合并型式有图76中所描述的树表示，其中我们称磁带为 A,B,C,D，以代替 T1，T2，T3，T4。

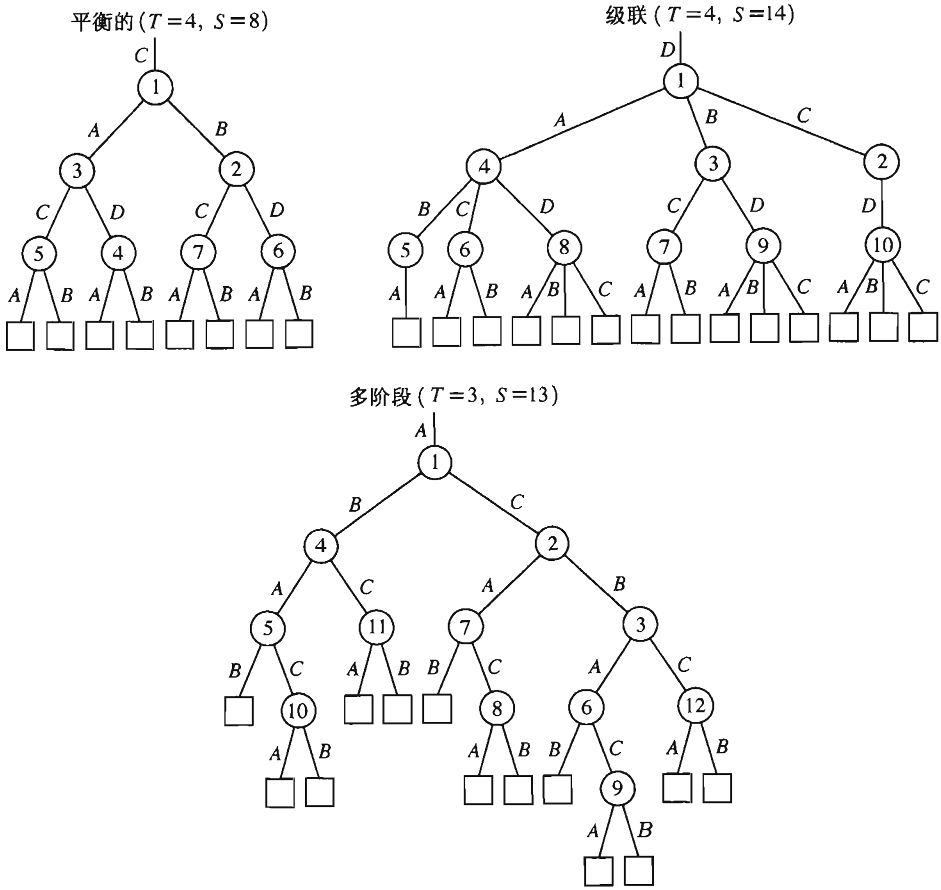


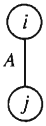
图76 3种合并型式的树表示

这个表示以方便的形式揭示了合并型式中许多有关的性质；例如，如果树的0级(根)上的路段是递增的，则1级上的路段必定是递降的，在2级上的那些必定是递增的，等等；一个初始路段是递增的，当且仅当对应的外部节点在一个偶数号级上。进而，在合并期间处理的初始路段的总数(不包括初始分布)恰等于这株树的外部路段长度，因为在 k 级上的每个初始路段都恰被处理 k 次。

每种合并型式都有一个树表示，但不是每个树都定义一种合并型式。一株其内部节点已经以数1到 m 标号，而其边已经以磁带名标号的树，表示一种正确的向后读的合并型式的充要条件为

a)凡邻接于同一内部节点的两边，磁带名都不相同。

b)如果 $i>j$,且如果 A 是一个磁带名,则这树不包含下列形状



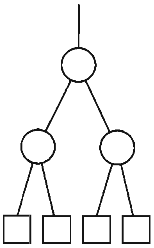
c)如果 $i<j<k<l$,而且如果 A 是一个磁带名,则这株树不包含下列形状



(4)

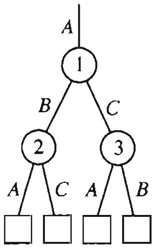
条件 a)是自明的,因为在一个合并中输入和输出的磁带必须是不同的;类似地,b)也是显然的。"无交叉"条件 c)反映了后进先出的限制,它表征了在磁带上向后读的操作:在步骤 k 中形成的一个路段,必须比同一磁带上以前形成的先撤销;因此(4)中构形是不可能的。不难验证,任何满足条件 a),b),c)的磁带标号的树确实对应于一种向后读的合并型式。

如果有 T 台磁带机,则条件 a)意味着每个内部节点的次数为 $T-1$ 或更小。对所有这样的树并不总能附加上适当的标号;例如,当 $T=3$ 时,没有其树的形状为



(5)

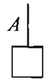
的合并型式。如果我们能以适当的方式附加步骤号和磁带名称,则这个形状将导致一个最优的合并型式,因为它是在有 4 个外部节点的一株树中达到极小外部路段长度的惟一途径。但由于图式的对称性,实质上仅有一种方式按照条件 a)和 b)来加标号,即



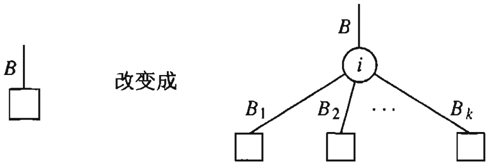
(6)

但这同条件 c)相冲突。可以按照上述条件,利用至多 T 个磁带名称加上标号的一种图形,称为 T-lifo(T-后进先出)树。

另外还有一种方式,可以刻划产生于合并型式的所有磁带标号树,即考虑怎样能“长出”所有这样的树。从某个磁带名称,比如说 A,以及以树秧



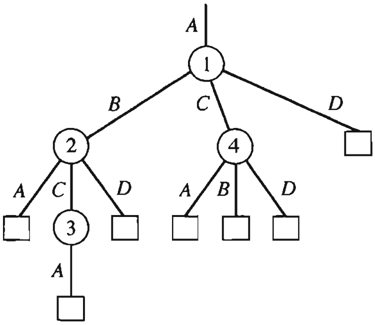
开始。在树生长过程中的第 i 步为选择不同的磁带名称 $B,B_1,B_2,\cdots,B_k$,并把对应于 B 的最近形成的外部节点



(7)

这个“最后形成,首先生长”的规则,精确地说明了树表示可以怎样从向量表示直接地构造出来。

要确定严格最优的 T-磁带合并型式,即从具有给定外部节点数的所有 T-lifo 树中选出具有极小路径长度的树,似乎是十分困难的。例如,下列并非显然的型式,可证明是向后读、合并 4 条磁带上 7 个初始路段的一种最优方式:



(8)

为达到最优,一个一路合并实际上是必不可少的(见习题 8)! 另一方面,对于任何固定的 T,要给出接近最优的构造,倒并不那么困难。

设 $K_T(n)$是在具有 n 个外部节点的一株 T-lifo 树中可达到的极小外部路径长度。由 2.3.4.5 小节中展示的理论,不难证明

$$K_T(n) \geqslant nq - \lfloor ((T-1)^q - n)/(T-2) \rfloor, \quad q = \lceil \log_{T-1} n \rceil \tag{9}$$

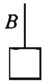
因为这是具有 n 个外部节点且所有节点的次数 $< T$ 的任何树的极小外部路径长度,现在确切地知道的 $K_T(n)$的值比较少。下面是部分上限,它们大概是准确的:

$n=$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
$K_3(n)\leqslant$	0	2	5	9	12	16	21	25	30	34	39	45	50	56	61	(10)
$K_4(n)\leqslant$	0	2	3	6	8	11	14	17	20	24	27	31	33	37	40	

Karp 已经发现，任何其内部节点的次数 $<T$ 的树都是准 T-lifo 的，就是说，只要改变某些外部节点成为一路合并，它就变成 T-lifo 的了。事实上，构造一组适当的标号是相当简单的。令 A 是一个特殊的磁带名称，则构造方法如下：

步骤 1. 假定特殊名称 A 仅用在一个分支的最左边的边上，则以同上述条件 a) 相一致的任何方式，把磁带名称附加到树图式的诸边上。

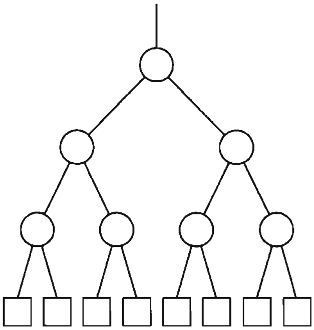
步骤 2. 每当 $B\neq A$ 时以 

代替形如 

的每一外部节点。

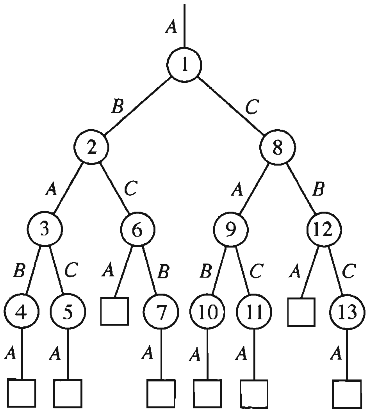
步骤 3. 按先序(preorder)为树的内部节点编号，结果就是满足条件 a),b),c)的一株磁带标号树。

例如，如果我们从树



(11)

和 3 条磁带开始，则这个过程可以赋标号如下：



(12)

不难验证,Karp 的构造满足"最后形成,最先生长"的规则,这是由于先序的本性所致(见习题 12)。

这个构造的结果是一个合并型式,它所有的初始路段都出现于磁带 A 上。它提示了以下的分布和排序方案,我们可以称之为先序合并。

P1. 分布初始路段到磁带 A 上,直到输入被穷尽为止。设 S 是初始路段的总数。

P2. 利用具有 S 个外部节点的一株极小路径长度的 $T-1$ 叉树,进行上述构造,得到一株其外部路径长度在(9)中下限的 S 倍之内的 T-lifo 树。

P3. 按照这一型式合并诸路径。▌

这个方案将在任何所希望的磁带上产生它的输出。但它有一个严重的缺陷——读者看出是什么了吗?它的问题在于合并型式要求开始时在磁带 A 上的某些路段是递增的,某些是递减的,这取决于对应的外部节点是出现在奇数级上还是出现在偶数级上。这个问题,通过在恰好需要之前,把应是递减的诸路段,拷贝到一条或多条辅助磁带上,就可无须预先知道 S 而解决。然后,用初始路段的长度来表达,处理的总数量为

$$S\ \log_{T-1}S + O(S) \tag{13}$$

于是,当 $S\to\infty$ 时,先序合并肯定好于多阶段或级联合并。其实,它是接近最优的,因为(9)表明,$S\ \log_{T-1}S+O(S)$是我们在 T 条磁带上所能希望达到的最好者。另一方面,对于在实践中通常出现的较小的 S 值,先序合并是相当低效的:当 S 相当小时,多阶段法或级联方法是更简单和更快的。也许有可能想出一种简单的分布和合并方案,对于小的 S 值,它足以同多阶段法和级联法匹敌,而对于大的 S,它是接近最优的。

以下的第二组习题表明 Karp 如何以一种类似的方式陈述了向前读合并的问题。在这种情况下,这个理论证明是更为复杂的,尽管已经发现了某些非常有趣的结果。

习　题——第一组

1.[*17*] 在向前读合并时,用一个键码为 $+\infty$ 的人造"哨兵"来标志磁带上每个路段的结尾往往是方便的。试问当向后读时,如何修改这一做法?

2.[*20*] 类似(1)的一个数组的诸列将总是非减的吗?或者,是否有这种可能,当我们从一级进行到下一级时,将要从某条磁带"减"去一些路段?

▶3.[*20*] 证明:如果 T1 原来从 *ADA*…开始,而 T2 到 T5 以 *DAD*…开始,则当时(1)的完全分布使用向后读多阶段合并后,在排序完成时,将总是在磁带 T1 上得到一个 *A* 路段。

4.[*M22*] 在以递增次序分布所有路段之后,做向后读多阶段合并是一个好的想法吗?想像所有的"*D*"位置开始时都填以虚拟路段。

▶5.[*23*] 当使用向后读多阶段合并时,合并数的序列将有什么公式,以代替 5.4.2 小节的(8),(9),(10)和(11)?通过画出类似图 71(a)的一个图形,说明在 6 条磁带上第 5 级分布的合并数。

6.[*07*] 树表示为(8)的合并型式,其向量表示如何?

7.[*16*] 画出由下列向量序列定义的向后读合并型式的树表示:

$v^{(33)}=(20,\ 9,\ 5)$	$y^{(22)}=(+1,-1,+1)$	$y^{(10)}=(+1,+1,-1)$
$y^{(33)}=(+1,-1,+1)$	$y^{(21)}=(-1,+1,+1)$	$y^{(9)}=(+1,-1,+1)$
$y^{(32)}=(+1,+1,-1)$	$y^{(20)}=(+1,+1,-1)$	$y^{(8)}=(+1,+1,-1)$
$y^{(31)}=(+1,+1,-1)$	$y^{(19)}=(-1,+1,+1)$	$y^{(7)}=(+1,+1,-1)$
$y^{(30)}=(+1,+1,-1)$	$y^{(18)}=(+1,+1,-1)$	$y^{(6)}=(+1,+1,-1)$
$y^{(29)}=(+1,-1,+1)$	$y^{(17)}=(+1,+1,-1)$	$y^{(5)}=(-1,+1,+1)$
$y^{(28)}=(-1,+1,+1)$	$y^{(16)}=(+1,+1,-1)$	$y^{(4)}=(+1,-1,+1)$
$y^{(27)}=(+1,-1,+1)$	$y^{(15)}=(+1,+1,-1)$	$y^{(3)}=(-1,+1,+1)$
$y^{(26)}=(+1,-1,+1)$	$y^{(14)}=(+1,-1,+1)$	$y^{(2)}=(+1,-1,+1)$
$y^{(25)}=(+1,+1,-1)$	$y^{(13)}=(+1,-1,+1)$	$y^{(1)}=(-1,+1,+1)$
$y^{(24)}=(+1,-1,+1)$	$y^{(12)}=(-1,+1,+1)$	$y^{(0)}=(1,\ 0,\ 0)$
$y^{(23)}=(+1,-1,+1)$	$y^{(11)}=(+1,+1,-1)$	

8.[*23*] 证明当 $S=7$ 和 $T=4$ 时,(8)是向后读合并的一种最优方式,那些避免一路合并的所有方法均比它差。

9.[*M22*] 证明下限(9)。

10.[*41*] 利用一台计算机,编制 $K_T(n)$ 的准确值的表。

▶11.[*20*] 真或假:对于只使用 $T-1$ 路合并的任何向后读合并型式,在每条磁带上只能有交替的 *ADAD*…的路段,如果两个相邻的路段以相同次序出现,则无法继续工作。

12.[*22*] 证明,Karp 的先序构造将总是产生一个满足条件 a),b)和 c)的磁带标号的树。

13.[*16*] 通过撤消尽可能多的一路合并,使得先序仍然给出内部节点的一个正确标号,从而使(12)更为有效。

14.[*40*] 试设计一个算法,它进行先序合并,而无须明显地表示在步骤 P2 和 P3 中的树,要求此算法仅仅用 $O(\log S)$个内存单元来控制合并型式。

15.[*M39*] 正文中的 Karp 先序构造，产生在若干终端节点处具有一路合并的树。证明当 $T=3$ 时，有可能构造接近最优的 3-lifo 树，其中自始至终使用两路合并。

换言之，设 $\hat{K}_T(n)$是所有的具有 n 个外部节点，且使得每个内部节点的次数为 $T-1$ 的 T-lifo树的极小外部路径长度，证明：$\hat{K}_3(n)=n\lg n+O(n)$。

16.[*M46*] 使用习题 15 的符号，当 $n\equiv 1$ (modulo $T-2$)时，是否对于所有的 $T\geqslant 3$，$\hat{K}_T(n)=n\log_{T-1}n+O(n)$？

▶17.[*28*] (Richard D. Pratt) 为了在向后读级联合并中实现递增次序，我们可坚持偶数次的合并扫描次数；这提示了有些不同于算法 5.4.3C 的初始分布技术。

a)改变 5.4.3-(1)，使得它仅仅示出要求偶数次合并扫描的完全分布。

b)试设计一初始分布方案，它内插于这些完全分布之间(于是，如果初始路段数落在几个完全分布之间，则希望两次合并某些路段，但不是所有的路段，以便达到一个完全的分布)。

▶18.[*M38*] 假设对于某个 $T\geqslant 3$，有 T 个磁带机可利用，而且 T1 含 N 个记录而其余的磁带为空。问是否有可能在少于 $\Omega(N\log N)$步内颠倒 T1 上记录的顺序，而无须向后读(当然，如果允许向后读，则操作是显然的)？对于要求阶为 $N\log N$ 步的这样一类算法，请参见习题 5.2.5-14。

习 题——第二组

下列习题发展了向前读带的磁带合并理论；在这种情况下，每条磁带起一个队列的作用，而不是一个栈的作用。一个合并型式可以完全像正文中那样，表示成向量 $y^{(m)}\cdots y^{(1)}y^{(0)}$的一个序列。但当我们把向量表示转换为一个树表示时，我们把“最后形成，首先生长”改为“首先形成，首先生长”，于是非法的形状(4)将改变为



(4′)

类似于向后读情况下的术语“T-lifo”，可以被标号，以便表示在 T 条磁带上的向前读合并的一株树，称为 T-fifo(T-先进先出)。

当磁带可以向后读时，它们形成非常好的栈。但不幸的是，它们并不形成非常好的通用的队列。如果以先进先出的方式，随机地写和读，则我们将浪费大量的时间把磁带移来移去。甚至更坏的是，可能会“跑出”磁带的末尾！我们会遇到与 2.2.2-(4)和(5)中队列超出内存同样的问题，但是 2.2.2-(6)和(7)中的解不能应用到磁带上，因为它们并不是圆形的循环。因此，如果我们可以对一株树加以标号，使得它对应的合并型式使每条磁带都遵循特殊的排队规则“写，重绕，读所有的记录，重绕；写，重绕，读所有的记录，重绕，等等”，则称这株树为强 T-fifo 的。

▶19.[*22*] (R.M.Karp) 试求一株不是 3-fifo 的二叉树。

▶20.[*22*] 利用类似于(4′)的关于非法的磁带标号形状的相当简单的规则，建立“强 T-fifo”的条件。

21.[*18*] 画出习题 7 中用向量定义的向前读合并型式的树表示，这株树是强 3-fifo 的吗？

22.[*28*] (R.M.Karp) 证明具有完全分布的多阶段和级联合并的树表示，除了标示内部节点的诸数外，对向后读及向前读两者的情况完全一样。试求更大一类合并型式的向量表示，对于它来说，上述条件为真。

23.[*24*] (R.M.Karp) 如果没有随后用作输入带的输出带，即：如果不存在 i,j,k，使 $q\geqslant i>k\geqslant r$，$y_j^{(i)}=-1$ 和 $y_j^{(k)}=+1$，则我们称一个合并型式的 $y^{(q)}\cdots y^{(r)}$段为一节(stage)。本题的目

的是证明，对有相同磁带数和初始路段数的所有合并型式，级联合并使数最小。

为叙述方便，我们先定义一些符号。如果 v 和 w 是 T 向量，使得存在一个合并型式，它在它的第一节把 w 约化为 v，则写之为 $v\to w$（于是存在一个合并型式 $y^{(m)}\cdots y^{(0)}$ 使得 $y^{(m)}\cdots y^{(l+1)}$ 为一节，$w=y^{(m)}+\cdots+y^{(0)}$ 且 $v=y^{(l)}+\cdots+y^{(0)}$）。如果 v 和 w 是 T 向量，使得对于 $1\leqslant k\leqslant T$，v 的最大 k 个元素之和 $\leqslant w$ 的最大 k 个元素之和，则写之为 $v\preceq w$。例如，$(2,1,2,2,2,1)\preceq(1,2,3,0,3,1)$，因为 $2\leqslant3$，$2+2\leqslant3+3$，$\cdots$，$2+2+2+2+1+1\leqslant3+3+2+1+1+0$。最后，如果 $v=(v_1,\cdots,v_T)$，则令 $C(v)=(s_T,s_{T-2},s_{T-3},\cdots,s_1,0)$，其中 s_k 是 v 的最大 k 个元素之和。

a)证明 $v\to C(v)$。

b)证明 $v\preceq w$ 蕴涵着 $C(v)\preceq C(w)$。

c)假定习题 24 的结果，证明级联合并使节的数目取极小。

24.[*M35*] 使用习题 23 的符号，证明 $v\to w$ 蕴涵着 $w\preceq C(v)$。

25.[*M36*] (R.M.Karp) 如果没有既用作输入也用作输出的磁带，即，如果不存在 i,j,k，使 $q\geqslant i\geqslant r$，$q\geqslant k\geqslant r$，且 $y_j^{(i)}=+1$ 和 $y_j^{(k)}=-1$，则我们称合并型式的一个 $y^{(q)}\cdots y^{(r)}$ 段是为一个阶段(phase)。本题的目的是研究使阶段数极小化的合并型式。如果在一个阶段中 w 可以被约化为 v（习题 23 中引进了类似符号），则写之为 $v\Rightarrow w$。我们设

$$D_k(v)=(s_k+t_{k+1},s_k+t_{k+2},\cdots,s_k+t_T,0,\cdots,0)$$

其中 t_j 表示 v 的第 j 个最大的元素，且 $s_k=t_1+\cdots+t_k$。

a) 证明对于 $1\leqslant k<T$，$v\Rightarrow D_k(v)$。

b) 证明对于 $1\leqslant k<T$，$v\preceq w$ 蕴涵着 $D_k(v)\preceq D_k(w)$。

c) 证明对于某个 k，$1\leqslant k<T$，$v\Rightarrow w$ 蕴涵着 $w\preceq D_k(v)$。

d) 因此，在 q 个阶段和 T 条磁带的情况下，能使被排序的初始路段数达到极大的一个合并型式，可以通过一个整数序列 $k_1k_2\cdots k_q$ 表示，使得初始分布是 $D_{k_q}(\cdots(D_{k_2}(D_{k_1}(u)))\cdots)$，其中 $u=(1,0,\cdots,0)$。这个极小阶段策略，有一个强 T-fifo 表示，而且它也属于习题 22 中型式的类型。当 $T=3$ 时，它就是多阶段合并。而对于 $T=4,5,6,7$，它是平衡合并的一种变种。

26.[*M46*] (R.M.Karp) 对于所有 $T\geqslant4$ 和所有充分大的 q，习题 25 中提到的最优序列 $k_1k_2\cdots k_q$ 总是等于 $1\lceil T/2\rceil\lfloor T/2\rfloor\lceil T/2\rceil\lfloor T/2\rfloor\cdots$ 吗？

*5.4.5 振荡排序

Sheldon Sobel 在 *JACM* **9**(1962)，372～375 中提出了一个与合并排序稍微不同的方法。这个方法不是以把所有初始路段分散到磁带的一个分布扫描开始，而是在分布和合并之间前后振荡，使得在对输入进行完备地考察之前，许多排序都可进行。

例如，假设有 5 条磁带可供合并时利用，Sobel 方法将 16 个初始路段排序如下：

	操作	T1	T2	T3	T4	T5	代价
阶段 1	分布	A_1	A_1	A_1	A_1	—	4
阶段 2	合并	—	—	—	—	D_4	4
阶段 3	分布	–	A_1	A_1	A_1	D_4A_1	4
阶段 4	合并	D_4	–	–	–	D_4	4
阶段 5	分布	D_4A_1	–	A_1	A_1	D_4A_1	4
阶段 6	合并	D_4	D_4	–	–	D_4	4

阶段 7	分布	D_4A_1	D_4A_1	–	A_1	D_4A_1	4
阶段 8	合并	D_4	D_4	D_4	–	D_4	4
阶段 9	合并	–	–	–	A_{16}	–	16

此处同 5.4.4 小节中一样，使用 A_r 和 D_r 分别代表相对长度为 r 的递增和递降路段。本方法开始在 4 条磁带上各写一个初始路段，并且把它们（向后读）合并到第 5 条磁带上。分布再次继续，这次磁带循环地向右移一个位置，而第二个合并产生另一个路段 D_4。在以这种方式形成 4 个 D_4 后，一个附加的合并建立 A_{16}。我们可以继续建立另 3 个 A_{16}，把它们合并成一个 D_{64}。如此类推直到穷尽输入为止。预先不必知道输入的长度。

当初始路段的个数 S 为 4^m 时，不难看出这个方法恰处理每个记录 $m+1$ 次：在分布期间 1 次和在合并期间 m 次。当 S 处于 4^{m-1} 和 4^m 之间时，我们可以假定存在虚拟路段，把 S 提升为 4^m；因此总共的排序时间实际上将等于对于所有的数据进行 $\lceil \log_4 S \rceil + 1$ 次扫描。这恰是通过对 8 条磁带的一个平衡排序所达到的；一般地说，使用 T 条工作磁带的交替排序等价于使用 $2(T-1)$ 条磁带的平衡的合并，因为它进行对数据的 $\lceil \log_{T-1} S \rceil + 1$ 次扫描。当 S 是 $T-1$ 的一个乘方时，这是任何 T-磁带方法所能达到的最好效果，因为它达到等式 5.4.4-(9) 中的下限。另一方面，当 S 是 $(T-1)^{m-1}+1$ 时，即恰巧比 $T-1$ 的一个乘方大 1 时，这个方法几乎浪费整个一趟扫描。

习题 2 示出，如何通过使用一个特殊的结尾例程，以部分消除对于非完整乘方的 S 的损失。1966 年，Dennes L. Bencher 做了进一步的改进，他称他的过程为“交叉合并”［见 H. Wedekind, *Datenorganisation* (Berlin: W. de Gruyter, 1970), 164～166; *U. S. Patent* 3540000(1970)］。主要的思想是延迟合并直到获得关于 S 的更多的知识为止。我们将讨论对 Bencher 创立的方案稍做修改后的形式。

这个改进的交替排序进行如下：

	操作	T1	T2	T3	T4	T5	代价
阶段 1	分布	—	A_1	A_1	A_1	A_1	4
阶段 2	分布	—	A_1	A_1A_1	A_1A_1	A_1A_1	3
阶段 3	合并	D_4	—	A_1	A_1	A_1	4
阶段 4	分布	D_4A_1	—	A_1	A_1A_1	A_1A_1	3
阶段 5	合并	D_4	D_4	—	A_1	A_1	4
阶段 6	分布	D_4A_1	D_4A_1	—	A_1	A_1A_1	3
阶段 7	合并	D_4	D_4	D_4	—	A_1	4
阶段 8	分布	D_4A_1	D_4A_1	D_4A_1	—	A_1	3
阶段 9	合并	D_4	D_4	D_4	D_4	—	4

这时我们不把 D_4 合并到 A_{16} 中（除非输入恰好将要穷尽），仅仅在实施

阶段 15	合并	D_4D_4	D_4D_4	D_4D_4	D_4	—	4

之后，我们将得到

阶段 16	合并	D_4	D_4	D_4	—	A_{16}	16

在再生成 3 个 D_4 之后，将出现第二个 A_{16}

阶段 22	合并	D_4D_4	D_4D_4	D_4	—	$A_{16}D_4$	4
阶段 23	合并	D_4	D_4	—	A_{16}	A_{16}	16

等等（参见阶段 1～5）。可以看出 Bencher 方案的优点，例如，如果仅有 5 个初始阶段，则在习题 2 中修改过的振荡排序将进行四路合并（在阶段 2），紧接着做一个二路合并，总共代价为 $4+4+1+5=14$。而 Bencher 方案则做一个二路合并（在阶段 3），紧接着做一个四路合并，总共代价为 $4+1+2+5=12$（两个方法都含一个小的附加代价，即在最后的合并之前有 1 个单位的重绕）。

以下的算法 B 中有 Bencher 方法的一个精确描述。可惜的是，它似乎是一个比代码更难以理解的过程。向一台计算机说明这项技术，比之于向一位计算机学家说明更容易！部分原因是由于它是一个递归方法，已经表达成迭代的形式，且进行了一定的优化；读者可以发现，要真正了解它，有必要跟踪整个算法若干次。

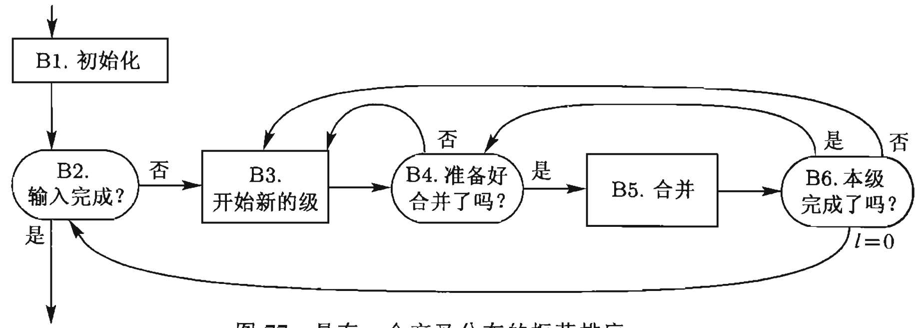


图 77 具有一个交叉分布的振荡排序

算法 B（*具有交叉分布的振荡排序*） 这个算法取初始路段并把它们分散到磁带上，偶尔中断分布过程以便合并某些磁带的内容。这个算法使用 p 路合并，并假定有 $T=P+1\geqslant 3$ 台磁带机可利用（用于保持输入数据的磁带机不计在内）。这些磁带机必须允许以向前和向后来进行读人，而且以数 $0,1,\cdots,p$ 标记。下列项是必需的：

$D[j], 0\leqslant j\leqslant P$ 假设出现在磁带 j 末端的虚拟路段号。

$A[l,j],\ 0\leqslant l\leqslant L, 0\leqslant j\leqslant P$ 这里 L 是使得至多有 P^{L+1} 个初始路段被输入的一个数。当 $A[l,j]=k\geqslant 0$ 时，名义上长度为 P^k 的一个路段出现在磁带 j 上，这对应于算法操作的“级 l”。如果 k 为偶，这个路段是递增的；如果 k 为奇，则是递降的。当 $A[l,j]<0$ 时，级 l 不使用磁带 j。

“写一初始路段到磁带 j 上”这一语句，为下列操作的简述：

置 $A[l,j]\leftarrow 0$。如果输入已穷尽，则 $D[j]$ 增 1；否则在磁带 j 上写一初始路段

(以递增的次序)。

"合并到磁带 j 上"这一语句为下列操作的简述:

如果对于所有 $i\neq j$,D[i]>0,则对所有 $i\neq j$,D[i]减 1 且 D[j]加 1。否则从所有使 D[i]=0 的磁带 i($i\neq j$)取一个路段合并到磁带 j 上,而且对于所有其它 $i\neq j$,D[i]减 1。

B1. [初始化] 对于 $0\leqslant j\leqslant P$,置 D[j]←0。置 A[0,0]←−1,l←0,q←0。然后对于 $1\leqslant j\leqslant P$,写一个初始路段到磁带 j 上。

B2. [输入完成?] (这时磁带 q 为空,而且其它磁带至多每条包含一个路段) 如果还有输入,则进行步骤 B3。但如果输入已穷尽,则重绕所有使得 A[0,j]为偶的磁带 $j\neq q$,然后在刚才重绕的磁带上向前读,而在其它的磁带上向后读,以此方式合并到磁带 q。这就完成了排序,且按递增次序排列的输出就在磁带 q 上。

B3. [开始新的级] 置 $l\leftarrow l+1$,$r\leftarrow q$,$s\leftarrow 0$ 且 $q\leftarrow(q+1)\bmod T$。对于 $1\leqslant j\leqslant T-2$,各写一初始路段到磁带$(q+j)\bmod T$ 上(于是在除磁带 q 和 r 之外的每条磁带上,各写一个初始路段)。置 A[l,q]←−1 和 A[l,r]←−2。

B4. [准备好合并了吗?] 如果 A[$l-1,q$]$\neq s$,则转回步骤 B3。

B5. [合并] (这时对所有的 $j\neq q$,$j\neq r$,A[$l-1,q$]=A[l,j]=s) 合并到磁带 r,并向后读(见上述此操作的定义)。然后置 $s\leftarrow s+1$,$l\leftarrow l-1$,A[l,r]←s 以及A[l,q]←−1。置 $r\leftarrow(2q-r)\bmod T$(一般说来,当 s 为偶时我们有 $r=(q-1)\bmod T$,当 s 为奇时 $r=(q+1)\bmod T$)。

B6. [本级完成了吗?] 如果 $l=0$,则转到 B2。否则如果对于所有 $j\neq q$ 和 $j\neq r$,A[l,j]=s,则转到 B4。否则返回 B3。 ▌

我们可以使用一个"递归归纳法"风格的证明来证明这个算法是正确的,正如对算法 2.3.1T 所做的那样。假设我们以 $l=l_0$,$q=q_0$,$s_+=$A[l_0,$(q_0+1)\bmod T$],以及 $s_-=$A[l_0,$(q_0-1)\bmod T$]开始步骤 B3;并进而假设 $s_+=0$,或 $s_-=1$,或 $s_+=2$,或 $s_-=3$,或……。用归纳法能够验证这个算法最终将达到 B5 而无须改变 A 的第 0 到第 l_0 诸行,而且 $l=l_0+1$,$q=q_0\pm 1$,$r=q_0$ 及 $s=s_+$ 或 s_-,这里如果 $s_+=0$ 或($s_+=2$ 或 $s_-\neq 1$)或($s_+=4$ 和 $s_-\neq 1,3$)或……时,我们选择正号+,而如果($s_-=1$ 和 $s_+\neq 0$)或($s_-=3$ 和 $s_+\neq 0,2$)或……时,我们选择负号−。这里简述的证明并不是最好的,因为这个算法本身就是以一种适合于实现而不是验证的方式来陈述的。

图 78 借助于每个记录作为初始路段数 S 的函数被合并的平均次数,示出了算法 B 的效率,其中假定初始路段都近似地等长(对于多阶段和级联排序,对应的图出现于图 70 和图 74 中)。在制作这张图时,已经使用了习题 3 中提到的一点改进。

基于我们在 5.4.4 小节中讨论的先序合并理论,R. M. Karp 建立了称做回旋排序(gyrating sort)的一个相关方法;参见由 Randall Rustin 编著的*Combinatorial Algo-*

rithms (Algorithmics Press,1972),21～29。

向前读　振荡排序型式看来要求向后读的能力,因为我们在合并新近输入的短路段时,也需要在某处存入长路段。然而,M. A. Goetz[*Proc. AFIPS Spring Joint. Comp. Conf.* **25**(1964),599～607]发现了一个仅仅使用向前读和简单的重绕,就能实现振荡排序的方法。他的方法同这一章中我们已经见到过的其它方案,在以下两方面有根本的不同:

a) 数据有时写在磁带的前端,同时知道在磁带中间的现存数据未被破坏。

b) 所有的初始字符串都有一固定的极大长度。

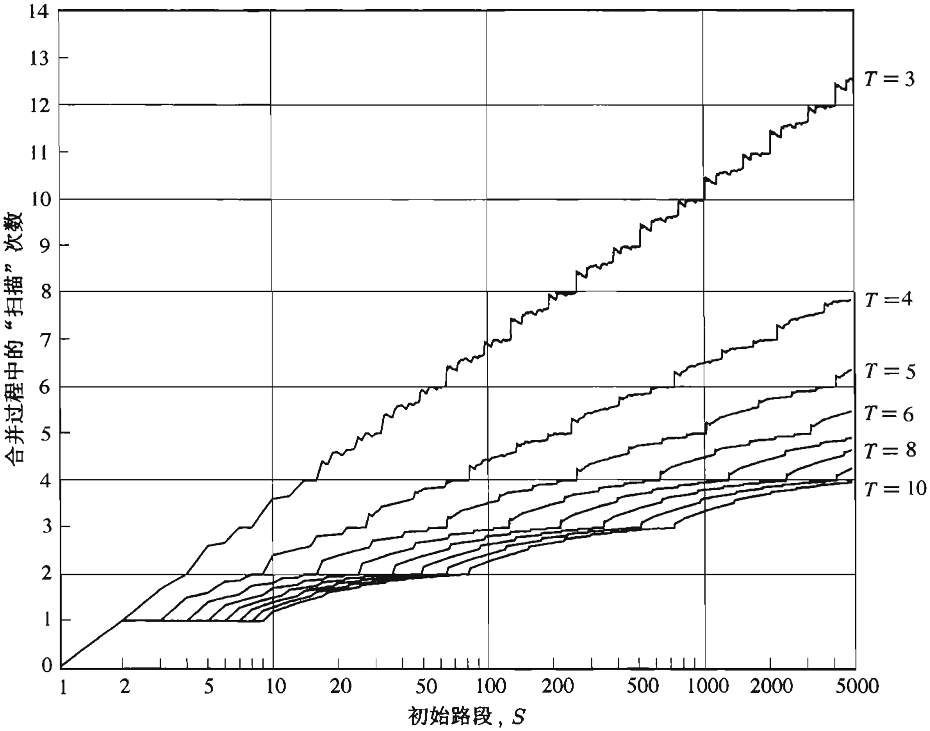


图 78　交替排序的效率,其中使用了算法 B 和习题 3 的技术

条件 a)违背了我们已经假定的、作为向前读特征的“-*fifo*”性质,但是只要在路段之间保留有充分数量的空白磁带而且在适当的时候忽略“奇偶错”的话,则这仍能可靠地实现。条件 b)同有效地使用替代选择看来不大相容。

作为一个算法,而不是一个物理设备来获得专利的最早算法之一[*U. S. Patent 3380029*(1968)],Goetz 的向前读振荡排序有稍微含糊的特性;除非成功地争辩,否则如果没有获得专利人的同意,这样一个专利就使得在一个程序中使用这个算法是非法的。几年后,IBM 支持 Bencher 的向后读振荡排序技术申请为专利[唉,现在我们只能说,发现一个算法的兴奋就足以令人满意,已经成了一个终结! 所幸的是,振

荡排序并非特别好。我们希望,发明最好算法的有公益心的人们,继续使他们的思想免费让人们使用。当然,那些把新技术完全保密的人,比起把算法的出现作为一个时间段内的个人财产者还要差得多]。

Goetz 方法的中心思想,是把事情安排成使得每条磁带以相对长度为 1 的一个路段开始,紧接着是相对长度为 P 的,然后相对长度为 P^2 的,等等。例如,当 $T=5$ 时,排序开始如下(利用“.”指出在每条磁带上当前读写头的位置):

操作	T1	T2	T3	T4	T5	代价	注释
阶段 1 分布	$.A_1$	$.A_1$	$.A_1$	$.A_1$	$A_1.$	5	[T5 尚未重绕]
阶段 2 合并	$\not{A}_1.$	$\not{A}_1.$	$\not{A}_1.$	$\not{A}_1.$	A_1A_4	4	[现在全都重绕]
阶段 3 分布	$.A_1$	$.A_1$	$.A_1$	$A_1.$	$.A_1A_4$	4	[T4 尚未重绕]
阶段 4 合并	$\not{A}_1.$	$\not{A}_1.$	$\not{A}_1$	$A_1A_4.$	$\not{A}_1.A_4$	4	[现在全都重绕]
阶段 5 分布	$A_1.$	$.A_1$	$A_1.$	$.A_1A_4.$	$.A_1A_4$	4	[T3 尚未重绕]
阶段 6 合并	$\not{A}_1.$	$\not{A}_1.$	$A_1A_4.$	$\not{A}_1.A_4$	$\not{A}_1.A_4$	4	[现在全都重绕]
阶段 7 分布	$.A_1$	$A_1.$	$.A_1A_4$	$.A_1A_4$	$.A_1A_4$	4	[T2 尚未重绕]
阶段 8 合并	$\not{A}_1.$	$A_1A_4.$	$\not{A}_1.A_4$	$\not{A}_1.A_4$	$\not{A}_1.A_4$	4	[现在全都重绕]
阶段 9 分布	$A_1.$	$.A_1A_4$	$.A_1A_4$	$.A_1A_4$	$.A_1A_4$	4	[T1 尚未重绕]
阶段 10 合并	$A_1A_4.$	$\not{A}_1.A_4$	$\not{A}_1.A_4$	$\not{A}_1.A_4$	$\not{A}_1.A_4$	4	[不重绕]
阶段 11 合并	$A_1A_4A_{16}.$	$\not{A}_1\not{A}_4.$	$\not{A}_1\not{A}_4.$	$\not{A}_1\not{A}_4.$	$\not{A}_1\not{A}_4.$	16	[现在全都重绕]

等等。在阶段 1,当 T2 接受它的输入时,T1 正被重绕,然后当 T3 接受它的输入时,T2 正被重绕,等等。最后,当输入穷尽时,虚拟路段将开始出现,有时有必要相像成它们已经以全长明显地写到磁带上。例如,如果 $S=18$,则在磁带 T4 和 T5 上的诸 A_1 在阶段 9 期间都将是虚拟的;当阶段 10 从 T2 到 T3 合并到 T1 时,在 T4 和 T5 上,我们将向前跳,因为我们必须转到 T4 和 T5 上的 A_4 那里,以便为阶段 11 做准备。另一方面,T1 上的虚拟路段 A_1 则不必明显地出现。因此,这个“压轴戏”有一点技巧。

下一节有此方法的另一个例子。

习 题

1.[22] 正文中说明了对于 $T=5$ 和 $S=16$ 的 Sobel 的最初的振荡排序。试给出一个算法的精确描述,该算法推广这个过程,对 $T=P+1\geqslant 3$ 磁带上的 $S=P^L$ 个初始路段进行排序,并力求简单。

2.[24] 在 Sobel 原先的方法中,如果 $S=6$,则我们可以假设 $S=16$ 并假设存在 11 个虚拟路段。这样在正文的例子中,阶段 3 将置虚拟路段 A_0 到 T4 和 T5 上,阶段 4 将把 T2 和 T3 上的 A_1 合并为 T1 上的一个 D_2。阶段 5~8 将不做什么;而阶段 9 将在 T4 上产生 A_6。更好的是在阶段 3 之后,就重绕 T2 和 T3,然后通过三路合并立即在 T4 上产生 A_6。

说明怎样来修改习题 1 的算法,使得当 S 不是 P 的完全乘方时,可得到与此类似的一个改进了的结果。

▶3.［29］假定有 9 个初始路段，编制一张说明 $T=3$ 时算法 B 之特性的图表。证明这个过程有一处显然是低效的，然后对算法 B 进行修正，以弥补这一情况。

4.［21］步骤 B3 置 A[l,q]和 A[l,r]为负的值。说明这两个情况之一总是多余的，因为对应的 A 表中的项是决不会被查找的。

5.［*M25*］设 S 是出现在算法 B 的输入中的初始路段数，S 的哪些值使得在步骤 B2 中不需要重绕？

5.4.6　关于磁带合并的实际考虑

麻烦的事情现在来了：至今我们已经讨论了各类合并型式，现在来看看它们如何真正地应用于计算机和磁带的实际配置，并且以有意义的方式来对它们进行比较。对于内部排序的研究表明，仅仅通过计算它执行的比较的数目，还不能适当地判断一个排序方法的效率；类似地，也不能仅仅通过知道它对于数据扫描的次数，就适当地评价一个外部排序方法。

在这一节里，将讨论典型磁带机的特征，及其对初始分布和合并的影响。特别是，我们将研究缓冲区分配的某些方案，及其对于运行时间的相应影响。我们也将简单地考虑排序生成器程序的构造。

磁带如何工作　不同厂家所提供磁带机的特征有很大不同。为方便起见，我们将定义一个假想的 MIXT 磁带机，在编写本书时，对于正被制造的设备来说，它是相当典型的。

MIXT 以 75in/s 的速度，读和写每英寸的 800 个字符。这意味着，当磁带活动时，每 1/60ms，或 $16\frac{2}{3}\mu s$，读或写一个字符。在 1970 年可以买到的实际的磁带机，其密度为每英寸 200 个字符到 1600 个字符，而其速度是从 $37\frac{1}{2}$in/s 到 150in/s，因此它们的有效速度是 MIXT 的 1/8 倍到 4 倍。

当然，在 5.4 节接近开始的地方我们就已经注意到，一般地说，磁带现在是非常过时了。但是在磁带排序曾经非常重要的那数十年间，我们曾经上了许多课，今天这些课仍然是有价值的。因此，在这里我们主要关心的并不是一些特定的答案，而是学习如何以合理的方式来把理论同实践结合起来。方法学要比现象学重要得多，因为不管技术如何变化，问题求解的原理都一直有用。如果读者暂时地把他们自己移植到 20 世纪 70 年代的思维方式中，那他们将会从这部分内容获得最大裨益。因此让我们假定我们仍然生活在那消逝了的年代吧！

以当时的观点来看，须要记住的重要事实之一就是，磁带有一个严格限定的容量。每卷磁带的长不超过 2400 英尺；因此，每一 MIXT 磁带卷至多可以容纳 23 000 000个字符左右，而读完全部这些字符要花费 23 000 000/3 600 000≈6.4min。如果必须对更大的文件排序，一般说来最好一次对一整卷进行排序，而后再把排好序的卷合并，以避免多余的磁带操作。这意味着，我们所研究过的实际出现在合并型式中的初始路段的个数 S，不可能是很大的。我们不可能找到 $S>5000$，即

使对于一个非常小的只产生 5000 个字符长的初始路段的内存也是如此。因此给出当 $S\to\infty$ 时算法的渐近有效性公式主要是在学术上有意义。

数据以块(block)出现在磁带上(见图 79),每个读/写指令传送一个块。块通常称为"记录",但我们将避免使用这一术语,因为它同"对一个由'记录'组成的文件排序"中的"记录"一词相冲突。在 50 年代所写的早期的许多排序程序中,这种区分是没有必要的,因为每个块写一个记录;但是我们将看到,在磁带上的每个块内多放几个记录通常是有利的。

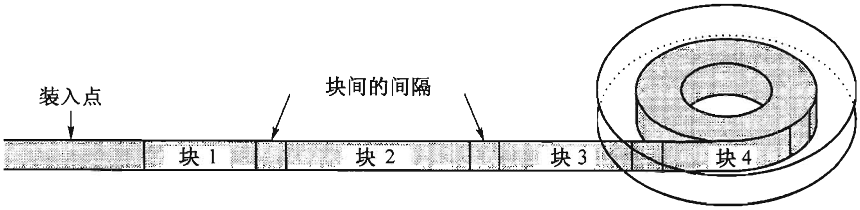


图 79 有可变大小的块的磁带

在相邻的块之间有一个 480 个字符位置长的块间间隔,为的是允许磁带在个别的读或写指令之间停止和启动。块间隔使得每卷磁带的字符数减少,其多少与每块的字符数相关(见图 80);同样,每秒传输的平均字符数也减少了,因为磁带是以相当固定的速度移动的。

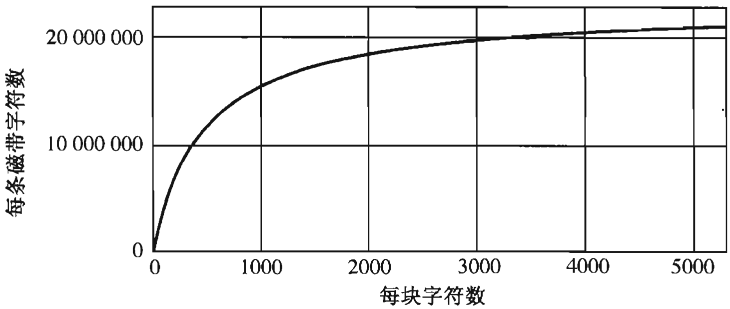


图 80 作为块大小的函数,每卷 MIXT 磁带的字符数

许多"旧式的"计算机只都有相当小的固定的块大小。如第一章中所定义的那样,它们的设计被反映在 MIX 计算机中,MIX 计算机总是读和写 100 字的块。这意味着每块大约 500 个字符,由于每个间隔为 480 个字符,因此几乎浪费一半的磁带!20 世纪 70 年代的大多数机器都允许块的大小可变,所以下面我们将讨论选择适当块大小的问题。

在一个读或写操作末了,磁带机以全速"滑过"间隔的头 66 个(左右的)字符。如果在这期间开始对同一条磁带做下一个操作,则磁带的运动无中断地继续下去。但如果下一个操作不能相当快地到来,则这条磁带将停止,而且它也需要一些时间

来加速到全速以进行下一个操作。停启时间的延迟加在一起为 5ms，停止 2ms 和启动 3ms（见图 81）。因此，如果我们错过了继续全速读的机会，则对运行时间的影响实质上和在块间隔中有 780 个字符（而不是 480 个字符）是一样的。

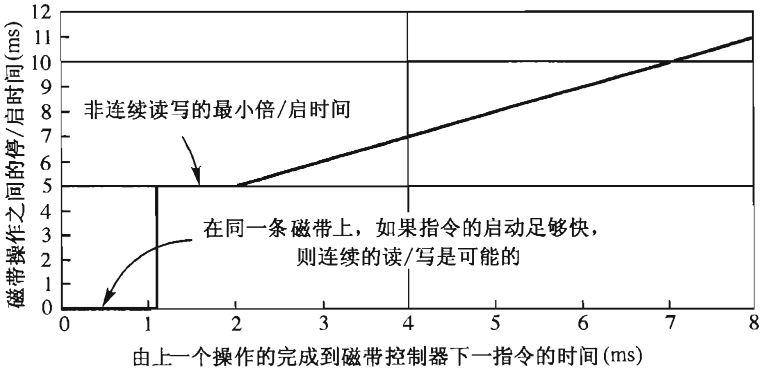


图 81　怎样计算停止/启动延迟时间
（加到用于读或写块及间隔的时间上）

现在我们考虑重绕操作。可惜，对于给定的字符数 n，一般难于表征重绕所需的确切时间。在某些机器上，有一个高速的重绕，它仅当 n 大于 5 百万左右时才可以应用；对于较小的 n 值，重绕则以通常的读/写速度进行。在其它机器上，有一个特殊的发动机用来控制所有的重绕操作；它逐渐地加快磁带卷达到每分钟某一转速，然后当其停止时即自动刹车。而实际的磁带速对于全卷都在变化。为了简便起见，我们将假定 MIXT 需要 max(30, n/150)ms 重绕 n 个字符位置（包括间隔在内），此即写它们所花时间的大约 3/5。这是对于许多实际磁带机行为的相当好的近似，在实际的磁带机中，读写时间同重绕时间之比一般都在 2 和 3 之间，但是这还未充分地模拟出在许多其它机器上出现的综合的低速和高速重绕的效果（见图 82）。

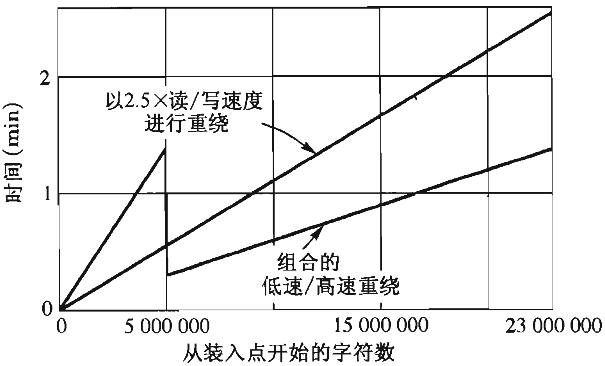


图 82　两个通常使用的重绕技术的近似运行时间

初始的装磁带和/或重绕都把一条磁带定位在“装入点”处，而在装入点处开始的任何读或写操作都需要额外的 110ms 的时间。当这条磁带不在装入点处时，可以向后读；在一个向前的操作之后的任何向后操作，或者在一个向后操作之后的任何向前操作，都需附加 32ms 的时间。

再论合并　现在让我们再次考察 P 路合并的过程，同时着重于考察输入和输出的活动，并假定 $P+1$ 台磁带机正在用于输入文件和输出文件。我们的目标是使输入/输出操作尽可能多地彼此重叠以及同程序的计算重叠，以使整个合并时间极小化。

考虑下列特殊情况是有益的：在这种情况下，对于可能的同时性设置了严格的限制。假设

a) 在任何一个时刻，至多一条磁带可写。

b) 在任何一个时刻，至多一条磁带可读。

c) 仅当读和写操作同时被启动后，读、写和计算可同时进行。

结果，尽管加上了这 3 个条件，$2P$ 个输入缓冲区和 2 个输出缓冲区的一个系统，就足以使磁带保持它实际上最快的速度来运行，除非计算机非常之慢。注意 a)实际上不是一个限制，因为仅有一条输出磁带。其次，输入的数量等于输出的数量，所以平均说来，在任何给定的时刻仅有一条磁带正在读；如果条件 b)不满足，则必定有一段时间没有输入。于是，如果我们保持输出磁带忙碌，则就能使合并的时间极小化。

一项称为预报(forecasting)的重要技术给了我们所希望的效果。在我们正在进行一个 P 路合并的同时，一般地有 P 个当前的输入缓冲区，它们被用作数据源；其中的某些比其余的更满些，这依赖于它们的数据已有多少被扫描。如果大约在同一时间它们全都变空了，则在可以往下进行之前，将需要大量地读，除非我们事先已经预见到这个意外事件。幸而，通过简单地考察每个缓冲区中的最后一个记录，总有可能知道哪个缓冲区将首先变空。最后记录有最小关键字的缓冲区将总是头一个变空，不管任何其它键码的值是什么。所以我们总是知道哪一个文件将是下一条输入指令的源文件。下列算法详细地叙述了这个原理。

算法 F(浮动缓冲区的预报)　本算法在 $P\geqslant 2$ 时控制长输入文件的 P 路合并期间的缓冲区安排。假定输入磁带和文件编号为 $1,2,\cdots,P$。这个算法使用 $2P$ 个输入缓冲区 I[1]，…，I[2P]；两个输出缓冲区 O[0]和 O[1]，以及下列辅助列表项：

A[j]，$1\leqslant j\leqslant 2P$：如果 I[j]可用作输入则为 0，否则为 1；

B[i]，$1\leqslant i\leqslant P$：包含迄今从文件 i 读入的最后块的缓冲区；

C[i]，$1\leqslant i\leqslant P$：当前用来存放来自文件 i 的输入的缓冲区；

L[i]，$1\leqslant i\leqslant P$：迄今从文件 i 读入的最后一个关键字；

S[j]，$1\leqslant j\leqslant 2P$：当 I[j]变为空时使用的缓冲区。

这里描述的算法并不终止，终止它的适当方式在以下讨论。

F1.［初始化］对于 $1\leqslant i\leqslant P$，从磁带 i 读第一个块到缓冲区 I[i]，置 A[i]$\leftarrow$1，A[$P+i$]$\leftarrow$0，B[i]$\leftarrow i$，C[i]$\leftarrow i$，并置 L[i]为在缓冲区 I[i]中最后一个记

录的键码，然后找 m 使得 L[m]=min{L(1),…,L(P)}；并置 $t \leftarrow 0$，$k \leftarrow P+1$，开始从磁带 m 读到缓冲区 I[k]。

F2.［合并］合并来自缓冲区 I[C[1]],…,I[C[P]]的记录到 O[t]，直到 O[t]满了为止。如果在这过程期间一个输入缓冲区，比如说 I[C[i]]变空而 O[t]仍未满，则置 A[C[i]]←0，C[i]←S[C[i]]，并继续合并。

F3.［完成输入/输出］等候直到以前的读（或读/写）操作完成。然后置 A[k]←1，S[B[m]]←k，B[m]←k，并置 L[m]成为 I[k]中最后记录的键码。

F4.［预报］求 m 使得 L[m]=min{L[1],…,L[P]}，并求 k 使得 A[k]=0。

F5.［读/写］开始从磁带 m 读到缓冲区 I[k]，并从缓冲区 O[t]写到输出磁带，然后置 $t \leftarrow 1-t$ 并返回到 F2。▌

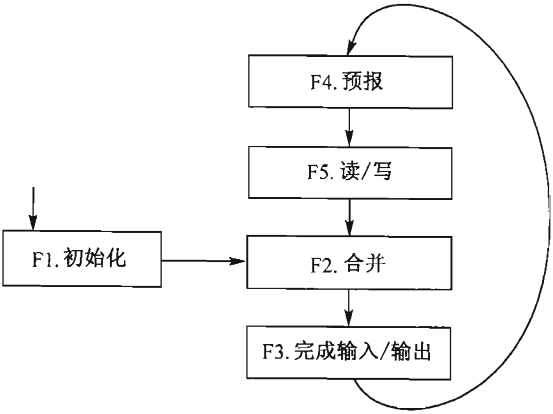


图 83　浮动缓冲区的预报

假定磁带上的每个块仅含两个记录，图 84 的例子示出当 $P=2$ 时，预报是如何进行的。所示的是每次我们达到步骤 F2 的开始处时输入缓冲区的内容，算法 F 实际上形成 P 个缓冲区队列，而以 C[i]指向第 i 个队列的前头，B[i]则指向队列尾。同时，S[j]指向缓冲区 I[j]的后继；这些指针如图 84 中的箭头所示，第 1 行说明了初始化之后的状态：对于每个输入文件有一个缓冲区，而另一个块正从文件 1 读入（因为 03<05）。第 2 行所示为在合并了第一个块之后的状态：我们正在输出包含“01　02”的一个块，并正在从文件 2 输入下一个块（因为 05<09）。注意，在第 3 行，4 个缓冲区中的 3 个实际上都被文件 2 占用，因为我们正从该文件读，而且在它的队例中已经有一个满缓冲区和一个部分满缓冲区。这种“浮动缓冲区”的安排是算法 F 的一个重要特性，因为如果选择文件 1 而不是文件 2 作为第 3 行的输入，那么我们就会无法进行到第 4 行。

为了证明算法 F 是正确的，必须证明两点：

i)总存在一个输入缓冲区可以利用（即在步骤 F4 中总能找到一个 k）。

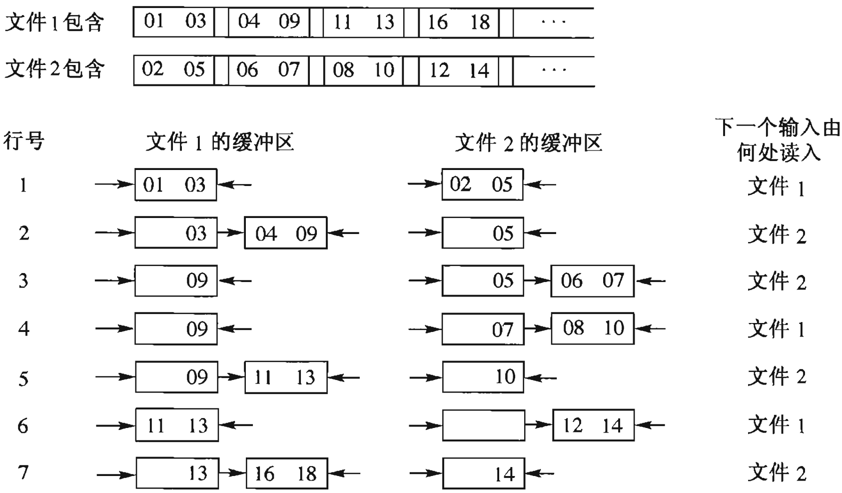


图 84 按照算法 F 的缓冲区队列

ii)如果在合并时一个输入缓冲区被穷尽,则它的后继者已经出现在内存中(即在步骤 F2 中的 S[C[i]]是有意义的)。

假设 i)为假,于是在我们到达步骤 F4 的某个时刻所有缓冲区都不能利用。每次到达该步骤时,在所有缓冲区中未处理的数据总数量恰恰是 P 个缓冲负载,即如果重新分布这些数据,则它们恰好足够填满 P 个缓冲区,因为我们正在以相同的速度输入和输出数据。某些缓冲区仅仅是部分地满了;但对于每个文件,至多有一个缓冲区是部分地满的,所以至多有 P 个缓冲区是部分地满的。由假设,所有 $2P$ 个缓冲区都不可利用,所以其中至少有 P 个必须是完全满的。这仅当 P 个是满的和 P 个是空的才会发生,否则我们将有太多的数据。但在任何一个时刻都至多只能有一个缓冲区是不可利用的和空的,因此 i)不能为假。

假设 ii)为假,于是我们在内存中没有对于某个文件的未处理的记录,但当前的输出缓冲区还不满。由预报原理,所有其它的文件不能有多于一个的数据块,因为在任何其它文件上的缓冲区都被穷尽以前,除非需要某个块时我们才读入该块。因此未处理的记录总共至多等于 $P-1$ 个块;加上未填满的输出缓冲区,将导出的内存中的数据量少于 P 个缓冲负载,矛盾。

这个论证确立了算法 F 的正确性,而且它也指出了出现病态情况的可能性,在这些病态情况下,本算法仅勉强地避免灾难。我们未曾提及的重要一点,即关于相等键码的可能性,在习题 5 中讨论。另外也请参见习题 4,它讨论了 $P=1$ 的情况。

如果刚刚读入的块是一个路段最后的块,则巧妙地结束算法 F 的一种方式,是在步骤 F3 中置 L[m]为∞(习惯上都以某种特殊的方式来指出一个路段的结束)。在读了所有文件上的所有数据之后,我们最终将发现在步骤 F4 中所有的 L 都等于∞;于是开始读每个文件上下一个路段的第一个块是可能的,并随着最后 $P+1$ 个块的输出,开始下一个合并阶段的初始化。

因此我们能使输出磁带实际上以全速运行，而无须在同一时间读一条以上的磁带。步骤 F1 中出现了这一规则的例外情形，这是为使所有东西都从开始处进行，所以一次读若干条磁带更有利。步骤 F1 通常都可安排成同计算的以前部分重叠。

考察每个块最后的记录，来预测哪一个缓冲区将首先变空的思想，是由 F. E. Holberton 于 1953 年发现的，而首先发表这项技术的则是 E. H. Friend[*JACM* **3** (1956)，144～145，165]。他的颇为复杂的算法使用 $3P$ 个输入缓冲区，每个输入文件专用 3 个缓冲区。算法 F 通过利用浮动缓冲区改善了这一点，允许任何一个文件一次要求多达 $P+1$ 个输入缓冲区，但总数决不多于 $2P$ 个。在这一节的末尾讨论了使用少于 $2P$ 个输入缓冲区的合并。在 5.4.9 节讨论了对算法 F 的某些有趣的改进。

合并型式的特性比较　现在让我们使用我们所掌握的关于磁带和合并的知识，来比较 5.4.2 小节到 5.4.5 小节中所研究的各种合并型式的有效性。当把每一种方法应用于相同的任务时，给出这些方法的细节是非常有益的。下面，我们就来考察一个文件的排序问题，该文件的每个记录包含 100 个字符，同时，内存中有 100 000个字符位置可以用来存储数据——不考虑程序和它的辅助变量，或者在一个选择树中由链接所占用的空间(记住我们仍假定在内存容量很小的那个时代)。输入以随机次序出现在磁带上，每块有 5 000 个字符，而输出以同样的格式出现。除了包含输入磁带的磁带机外，还有 5 条“空白磁带”可用。

有待排序的记录总数为 100 000，但排序算法事先并不知道这个信息。

图表 A 综述了把 10 个不同的合并方案应用到这批数据时发生的动作。考察这个重要图示的最好方式是想像你正在观看排序的进行：从左到右缓慢地扫描每行，假想你真正能看 6 条磁带的读、写、重绕和/或向后读，如同图中所指出的那样。在 P 路合并期间，输入磁带被移动的次数仅仅是输出磁带移动次数的 $1/P$。当原来的输入磁带已经完全读入(又已重绕和“锁磁带”)时，图 A 假定一个熟练的计算机操作员仅在 30s 内即卸下它并且以一条空白磁带代替。在例 2，例 3 和例 4 中，这是计算机空闲地等候操作员工作的“关键通路时间”；但在余下的例子中，卸磁带和重装操作是同其它处理重叠的。

例 1　向前读的平衡合并　让我们回顾这个问题的说明，记录的长度是 100 个字符，在每个时刻有足够的内存来保持 1000 个记录，而且在输入磁带上每个块包含 5000 个字符(50 个记录)。全部共有 100 000 个记录(＝10 000 000 字符＝2000 个块)。

对于中间文件我们自由地选择块的大小，一个 6 磁带的平衡的合并使用三路合并，所以算法 F 的技术要求 8 个缓冲区，我们因此可以使用每个含 1000/8＝125 个记录(＝12 500 个字符)的块。

初始分布扫描可以利用替换选择(算法 5.4.1R)，而且为了保持磁带光滑地运行，我们可以使用每个有 50 个记录的两个输入缓冲区，加上每个有 125 个记录的两个输出缓冲区，这就在替换选择树中保留了 650 个记录的空间。因此大多数初始路

段大约将有 1300 个记录长(10 或 11 个块),结果在图表 A 中产生了 78 个初始路段,最后一个要短些。

上述第一趟合并扫描中有 9 个路段合并到磁带 4,而不是在磁带 4,磁带 5 和磁带 6 之间交替进行。这使得在计算机操作员装入一条空白磁带到第 6 台磁带机上的同时有可能来做有用的工作;因为一旦已经完成了初始分布就知道了路段的总数,这个算法知道$\lceil S/9\rceil$个路段应该被合并到磁带 4,然后$\lceil (S-3)/9\rceil$个到磁带 5,然后$\lceil (S-6)/9\rceil$个到磁带 6。

利用 5.4.2 小节引进的符号,对于这个例子的整个排序过程可以下列方式概括如下:

1^{26}	1^{26}	1^{26}	—	—	—
—	—	—	3^9	3^9	3^8
9^3	9^3	$9^2 6^1$	—	—	—
—	—	—	27^1	27^1	24^1
78^1	—	—	—	—	—

例 2 向前读多阶段合并 图表 A(见书末)中的第二个例子按照算法 5.4.2D 进行多阶段合并。在此情况下,我们进行五路合并,使得存储器分成为每个有 83 个记录的 12 个缓冲区。在初始替换选择期间,我们有两个各 50 个记录的输入缓冲区和两个各 83 个记录的输出缓冲区,在树中保留 734 个记录;所以这次初始路段的长大约是 1468 个记录(17 或 18 个块)。所示的状态表明得到 $S=70$ 个初始路段,最后两个实际上分别仅有 4 个块和 1 个块长。合并型式可概括为

$0^{13}1^{18}$	$0^{13}1^{17}$	$0^{13}1^{15}$	$0^{12}1^{12}$	$0^8 1^8$	–
1^{15}	1^{14}	1^{12}	1^8	–	$0^8 1^4 2^1 5^3$
1^7	1^6	1^4	–	4^8	$1^4 2^1 5^3$
1^3	1^2	–	8^4	4^4	$2^1 5^3$
1^1	–	$16^1 19^1$	8^2	4^2	5^2
–	34^1	19^1	8^1	4^1	5^1
70^1	–	–	–	–	–

多奇怪,多阶段实际上比不甚复杂的平衡合并多花大约 25s! 关于此,主要有两个原因:

1)平衡合并在这种情况下特别走运,因为 $S=78$ 如此接近于 3 的一个完全乘方。如果产生的是 82 个初始路段,则平衡合并将要花费一趟额外的扫描。

2)多阶段合并在换输入磁带时浪费了 30s,而且在它等候重绕操作完成的同时总共花去 5min 以上。对比之下平衡合并需要比较少的重绕时间。在多阶段合并的第二个阶段,由于可假定磁带 6 上的 8 个虚拟路段在该磁带重绕时即存在,因此节省了 13s,但是并不出现其它的重绕重叠。因此,多阶段法失败了,尽管它要求相当少的读写时间。

例 3 向前读级联合并 此例情况类似于前例,但是使用算法 5.4.3C。合并过程可概括如下:

1^{14}	1^{15}	1^{12}	1^{14}	1^{15}	—
1^{5}	1^{9}	—	1^{14}	1^{15}	$1^{3}2^{3}3^{6}$
$5^{1}6^{3}$	5^{3}	$5^{3}6^{2}$	—	1^{1}	2^{2}
—	12^{1}	6^{1}	18^{1}	18^{1}	16^{1}
70^{1}	—	—	—	—	—

记住,通过浏览图表 A,观看每一个例子的进行情况。

例 4 分磁带的多阶段合并 在 5.4.2 小节末尾描述的这一过程,允许重叠大多数重绕时间。它使用四路合并,所以我们把内存分成 10 个 100 个记录的缓冲区;在替换选择树中有 700 个记录,所以结果形成了 72 个初始路段。最后的路段又是非常短的。使用了一个类似于算法 5.4.2D 的分布方案,之后接着一简单但稍微特别的设置虚拟路段的方法:

1^{21}	1^{19}	1^{15}	1^{8}	—	$0^{2}1^{9}$
$0^{2}1^{17}$	$0^{2}1^{15}$	$0^{2}1^{11}$	$0^{2}1^{14}$	—	$0^{2}1^{9}4^{4}$
1^{13}	1^{11}	1^{7}	—	$0^{2}4^{4}$	$0^{2}1^{9}4^{4}$
1^{10}	1^{8}	1^{4}	—	$0^{2}4^{4}3^{2}4^{1}$	$1^{8}4^{4}$
1^{6}	1^{4}	—	4^{4}	$0^{2}4^{4}3^{2}4^{1}$	$1^{4}4^{4}$
1^{5}	1^{3}	—	$4^{4}3^{1}$	$0^{1}4^{4}3^{2}4^{1}$	$1^{3}4^{4}$
1^{2}	—	$3^{1}7^{2}$	$4^{4}3^{1}$	$4^{2}3^{2}4^{1}$	4^{4}
1^{1}	—	$3^{1}7^{2}13^{1}$	$4^{3}3^{1}$	$4^{1}3^{2}4^{1}$	4^{3}
—	13^{1}	$3^{1}7^{2}13^{1}$	$4^{2}3^{1}$	$3^{2}4^{1}$	4^{2}
—	$13^{1}14^{1}$	$7^{2}13^{1}$	$4^{1}3^{1}$	$3^{1}4^{1}$	4^{1}
18^{1}	$13^{1}14^{1}$	$7^{1}13^{1}$	3^{1}	4^{1}	—
18^{1}	14^{1}	13^{1}	—	—	27^{1}
—	—	—	72^{1}	—	—

结果表明,这是图表 A 中所有不向后读的例子中最好的运行时间。因为 S 绝不会达到非常之大,故有可能研究出一个更为复杂的算法,它以一个甚至更好的方式来设置虚拟路段(参见等式 5.4.2-(26))。

例 5 具有重绕重叠的级联合并 尽管支配这个过程的算法要简单得多,但它却几乎像前面的例子一样快地运行。对于初始分布,我们像在算法 5.4.3C 中那样简单地使用级联排序方法,但用 $T=5$ 来代替 $T=6$。然后每个"级联"的每个阶段错开磁带的运行,使得我们通常不写磁带,除非它已经得到机会被重绕过了。这个型式可简略地写为:

1^{21}	1^{22}	1^{19}	1^{10}	—	—
1^{4}	1^{7}	—	—	$1^{2}2^{2}3^{5}$	4^{10}
7^{2}	—	8^{3}	$7^{2}8^{2}$	—	4^{1}
—	26^{1}	—	8^{1}	22^{1}	16^{1}
72^{1}	—	—	—	—	—

例 6 向后读的平衡合并 此例类似例 1,但消去了所有的重绕:

A_1^{26}	A_1^{26}	A_1^{26}	—	—	—
—	—	—	D_3^9	D_3^9	D_3^8
A_9^3	A_9^3	$A_9^2A_6^1$	—	—	—
—	—	—	D_{24}^1	D_{27}^1	D_{27}^1
A_{78}^1	—	—	—	—	—

由于例 1 中重绕比较少,这个方案不比向前读的情况好很多。事实上,尽管 $S=78$ 是较好的值,但它实际比分磁带的多阶段稍慢。

例 7 向后读多阶段合并 在这个例子中,仅使用了 6 条磁带中的 5 条,为的是消去重绕时间和更换输入磁带。于是,仅仅使用了四路合并,而且缓冲区的分配就像例 4 和例 5 那样。使用了类似算法 5.4.2D 那样的分布,但路段的方向是交替的,磁带 1 固定为最后的输出磁带。首先把一个递增的路段写到磁带 1 上;然后把递减的路段写到磁带 2,3,4 上;然后把递增的路段写到磁带 2,3,4 上;然后把递减的路段写到磁带 1,2,3 上等等。每次转换方向时,替换选择通常产生一个更短的路段,所以结果形成了 77 个初始路段,而不是例 4 和例 5 中的 72 个。

这个过程得到了(22,21,19,15)个路段的一个分布,而下一个完全的分布为(29,56,52,44)。习题 5.4.4-5 注明了怎样生成合并数串,以使生成的数串可以用来把虚拟路段放置在最优的位置上;由于一个磁带卷的有限性,确保了 S 绝不太大,所以这样一个过程在实际上是可行的。因此,图表 A 中的例子就是用这种设置虚拟路段的方法构造的(见习题 7),在所有图示例子中这证明是最快的。

例 8 向后读级联合并 如同例 7 中那样,此例仅仅使用 5 条磁带。这个过程遵循算法 5.4.3C,使用重绕和向前读来避免一路合并(因为重绕在 MIXT 磁带机上比读入要快两倍以上)。所以分布和例 6 一样。这个型式可以简单地概括如下,使用↓来表示重绕:

A_1^{21}	A_1^{22}	A_1^{19}	A_1^{10}	—
$A_1^4\downarrow$	$A_1^7\downarrow$	—	$D_1^2D_2^2D_3^5$	D_4^{10}
$A_8A_7^2$	A_5^2	A_9^4	—	$D_4^1\downarrow$
—	D_{17}	$A_9\downarrow$	D_{25}	D_{21}
A_{72}	—	—	—	—

例 9 向后读振荡排序 对于 $T=5$ 的振荡排序(算法 5.4.5B),可以使用如例 4,例 5,例 7 和例 8 那样的缓冲区分配,因为它进行四路合并。然而,替换选择不以同样的方式进行,因为恰在进入每个合并阶段之前要输出一个长度为 700 的(而不是 1400 左右的)路段,为的是清内存。因此,在此例中,产生 85 个路段,而不是 72 个。在这个过程中的某些关键步骤为

—	A_1	A_1A_1	A_1A_1	A_1A_1
D_4	—	A_1	A_1	A_1

D_4D_4	D_4D_4	D_4D_4	D_4	—
D_4	D_4	D_4	—	A_{16}

D_4	$A_{16}D_4D_4$	$A_{16}D_4$	$A_{16}D_4A_1$	A_{16}
D_4	$A_{16}D_4D_4$	$A_{16}D_4D_1$	$A_{16}D_4$	A_{16}
—	$A_{16}D_4$	$A_{16}D_4$	A_{16}	$A_{16}A_{13}$
—	$A_{16}D_4$	A_{16}	$A_{16}A_4$	$A_{16}A_{13}$
—	A_{16}	$A_{16}A_4$	$A_{16}A_4$	$A_{16}A_{13}$
D_{37}	—	$A_{16}\downarrow$	$A_{16}\downarrow$	$A_{16}\downarrow$
—	A_{85}	—	—	—

例 10　向前读振荡排序　在此最后的例子中,不使用替代选择,因为所有的初始路段都必须是相同长度的。因此每当需要一个初始路段时,占满整个内存的1000 个记录被内部排序;这使得 $S=100$。在这过程中某些关键步骤为

A_1	A_1	A_1	A_1	A_1
—	—	—	—	A_1A_4

A_1	A_1	A_1	A_1	A_1A_4
—	—	—	A_1A_4	$\not{A}_1A_4$
—	—	—	A_1A_4	A_1A_4

A_1	A_1A_4	A_1A_4	A_1A_4	A_1A_4
A_1A_4	$\not{A}_1A_4$	$\not{A}_1A_4$	$\not{A}_1A_4$	$\not{A}_1A_4$
$A_1A_4A_{16}$	—	—	—	—

—	A_1A_4	A_1A_4	A_1A_4	$A_1A_4A_{16}A_{64}$
A_4	$\not{A}_1A_4$	$\not{A}_1A_4$	$\not{A}_1A_4$	$\not{A}_1A_4A_{16}A_{64}$
A_4A_{16}	—	—	—	$\not{A}_1\not{A}_4A_{16}A_{64}$
$\not{A}_4A_{16}$	A_4	—	—	$\not{A}_1\not{A}_4A_{16}A_{64}$
—	—	—	A_{36}	$\not{A}_1\not{A}_4\not{A}_{16}A_{64}$
A_{100}	—	—	—	—

这个例程是所有程序中最慢的,部分原因是未使用替换选择,但主要是因为它的结尾相当笨拙(一个两路合并)。

估计运行时间　下面让我们来看看如何算出使用 MIXT 磁带的排序方法的近似执行时间。如果不进行详细的模拟,我们是否也能预测图表 A 中的结果呢?

用来比较不同合并型式的传统方法之一是叠印图,如图 70,图 74 和图 78 中所

示的那样。在假定每个初始路段的长度大抵相同的条件下，这些图给出作为初始路段个数的函数对数据的有效扫描次数（见图 85）。但这并不是非常现实的比较，因为我们已经看到，不同的方法导致不同数目的初始路段；其次，由于块间隔的疏密不同，引起开销时间也不同，而且重绕时间也有相当大的影响。所有这些同机器有关的特性，使我们不可能编制出独立于机器的对诸方法进行正确比较的图表。另一方面，图 85 说明了，除了平衡的合并之外，扫描的有效次数可以通过形如 $\alpha \ln S+\beta$ 的光滑曲线相当好地逼近。因此，在任何特定的情况下，都可以通过研究运行时间的近似公式，来对各个方法进行相当好的比较。我们的目标当然是要找出简单而且非常实用的公式来。

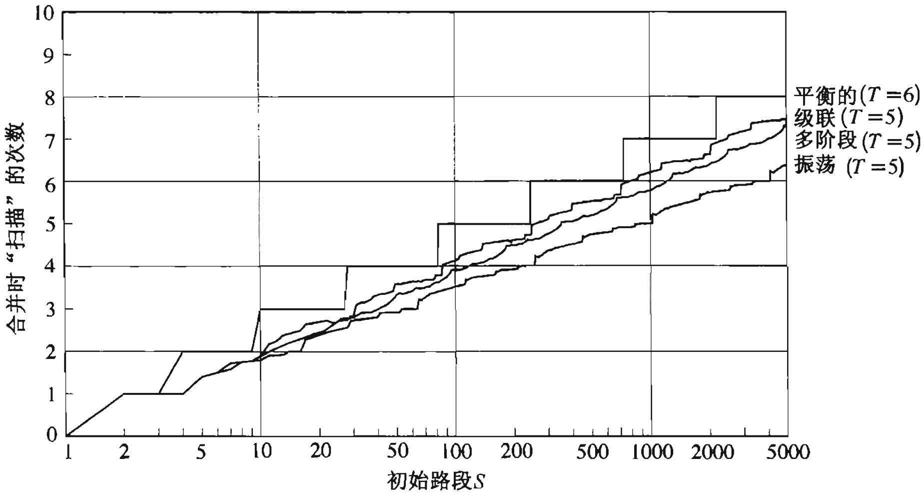


图 85　比较合并型式的一种欠妥的方法

现在我们借助于下列参数，尝试建立这样的一些公式：

N = 有待排序的记录数；

C = 每个记录的字符数；

M = 在内存中可以利用的字符位置数（假定为 C 的倍数）；

τ = 读或写一个字符的秒数；

$\rho\tau$ = 重绕一个字符的秒数；

$\sigma\tau$ = 停止/启动时间延迟的秒数；

γ = 每个块间隔的字符数；

δ = 操作员为卸掉和替换输入磁带所需秒数；

B_i = 在未排序的输入中每个块的字符数；

B_0 = 在排好序的输出中每个块的字符数。

对于 MIXT 我们有 $\tau=1/60000$，$\rho=2/5$，$\sigma=300$，$\gamma=480$。上面讨论的应用例子有 $N=100000$，$C=100$，$M=100000$，$\delta=30$，$B_i=B_0=5000$。这些参数通常都是影

响排序时间的机器和数据特征中最关键的(尽管重绕时间通常都是以比较复杂的一个表达式给出的,而不只是一个简单的比率 ρ)。给定了上述参数和一个合并型式,我们将进一步计算诸如下面的一些量:

P = 在此型式中合并的极大阶数;

P' = 替代选择树中的记录个数;

S = 初始路段个数;

$\pi = \alpha \ln S + \beta$ = 读和写每个字符的近似平均次数,未计入初始分布或最后合并;

$\pi' = \alpha' \ln S + \beta'$ = 在中间合并阶段对每个字符进行重绕的近似平均次数;

B = 在中间合并阶段每个块的字符数;

$\omega_i, \omega, \omega_0$ = “开销比”,读或写一个字符所需有效时间(由于间隔和停止/启动)除以硬件的时间 τ。

图表 A 的例子,已经根据公式

$$B = \left\lfloor \frac{M}{C(2P+2)} \right\rfloor C \tag{1}$$

选择了块和缓冲区的大小,使得在同算法 F 的缓冲方案相一致的前提下块尽可能地大(为了避免最后扫描期间的麻烦,P 应该足够小,使得(1)能使 $B \geqslant B_0$)。因此,在替代选择期间,树的大小为

$$P' = (M - 2B_i - 2B)/C \tag{2}$$

对于随机的数据,利用 5.4.1 小节的结果,初始路段的个数 S 可以估计为

$$S \approx \left\lceil \frac{N}{2P'} + \frac{7}{6} \right\rceil \tag{3}$$

假定 $B_i < B$,以及在分布期间输入磁带可以以全速进行(见下文),则大约花费 $NC\omega_i\tau$ 秒来分布初始路段,其中

$$\omega_i = (B_i + \gamma)/B_i \tag{4}$$

在合并时,缓冲方案允许同时读、写和计算,但输入磁带之间频繁的转换意味着必须增加由停止/启动磁带来的时间花费;因此我们置

$$\omega = (B + \gamma + \sigma)/B \tag{5}$$

而合并时间近似于

$$(\pi + \rho\pi')NC\omega\tau \tag{6}$$

这个公式稍微地惩罚了重绕时间,因为 ω 包括了停止/启动时间,但其它的考虑,诸如重绕的互锁及从装磁带点起读造成的额外花费,通常都能对此做出补偿。假定 $B_0 \leqslant B$,则最后的合并扫描是通过开销比

$$\omega_0 = (B_0 + \gamma)/B_0 \tag{7}$$

来加以限制的。

我们可以估计最后的合并和重绕的运行时间为

$$nC(1+\rho)\omega_0\tau$$

在实践中,由于存在不相等的块长度,它可能要花稍微长些的时间(输入和输出不像

在算法 F 中那样同步),但运行时间对于所有的合并型式都将大致相同。

在讨论对于个别型式的更特定的公式之前,让我们先来试着论证上述所做的两个假定。

a)替代选择能够跟上输入磁带吗? 在图表 A 的示例中,它基本能够,因为它大约花费算法 5.4.1R 的内循环的 10 次迭代来选择下一个记录,而且我们有 $C\omega_i\tau > 1667\mu s$ 来做这件事。只要在编写替代选择循环的程序时多加斟酌,在许多机器上(即使在 20 世纪 70 年代)都是做得到的。注意,这一情况在合并时不是太关键,因为每个记录的计算时间,几乎总是小于在一个 P 路合并期间每个记录的磁带运行时间,因为 P 不太大。

b)如(1)中那样,我们果真应选择 B 为缓冲区的最大容量吗? 一个很大的缓冲区会降低(5)中的开销比 ω,但它也会增加内部路段 S 的个数,因为 P'减小了。哪一个因素更重要,并不是一眼就能看出来的。把合并时间看成 $x = CP'$的函数,我们可以把它近似表示成

$$\left(\theta_1\ln\left(\frac{N}{x} + \frac{7}{6}\right) + \theta_2\right)\left(\frac{\theta_3 - x}{\theta_4 - x}\right) \tag{8}$$

其中 $\theta_1, \theta_2, \theta_3, \theta_4$ 是某些适当的常数,且 $\theta_3 > \theta_4$。关于 x 求微商,即可看出存在某个 N_0,使得对于所有 $N \geqslant N_0$,牺牲缓冲区的大小来增加 x 并不值得。例如,在图表 A 的排序应用中,N_0 大约等于 10 000;当对多于 10 000 个记录排序时,大的缓冲区是有利的。

然而要注意,当 S 超过 P 的一个乘方时,平衡合并的扫描次数急剧跃升。如果预先知道 N 的一个近似,则缓冲区大小应该被选择成使得 S 尽可能地稍小于 P 的一个乘方。例如,图表 A 第一行的缓冲区大小为 12 500;由于 $S = 78$,这是非常令人满意的,但如果 S 是 82,则减少一点缓冲区大小将要好得多。

10 个示例的公式 回到图表 A,让我们试着给出一些公式,这些公式近似为 10 个方法中每个的运行时间。在大多数情况下,只要我们确定了中间合并扫描的次数 $\pi = \alpha\ln S + \beta$ 和中间重绕扫描的次数 $\pi' = \alpha'\ln S + \beta'$,则基本的公式

$$NC\omega_i\tau + (\pi + \rho\pi')NC\omega\tau + (1 + \rho)NC\omega_0\tau \tag{9}$$

将是对整个排序时间的充分好的近似。对(9)有时有必要附加进一步的校正。每一个方法的细节可以叙述如下:

例 1 向前读平衡合并 对于在 $2P$ 条磁带上的 P 路合并,可以使用公式 $\pi = \lceil \ln S/\ln P\rceil - 1$, $\pi' = \lceil \ln S/\ln P\rceil/P$。

例 2 向前读多阶段合并 我们可以取 $\pi' \approx \pi$,因为每个阶段的后面通常都跟有大约和以前的合并长度相同的重绕。由表 5.4.2-1 我们得到,在 6 条磁带的情况下值 $\alpha \approx 0.795$, $\beta \approx 0.864 - 2$(减 2 是由于表中的项除了中间的扫描外,还包括初始和最后的扫描)。在初始分布之后重绕输入磁带的时间,即 $\rho NC\omega_i\tau + \delta$,应该加到(9)上。

例 3　向前读级联合并　表 5.4.3-1 给出了值 $\alpha\approx0.773,\beta\approx0.808-2$。估计重绕时间比较困难；或许置 $\pi'\approx\pi$ 就足够精确了。如同例 2 一样，我们需加初始重绕时间到(9)上。

例 4　分磁带多阶段合并　表 5.4.2-6 给出 $\alpha\approx0.752,\beta\approx1.024-2$。除开初始化($\rho NC\omega_i\tau+\delta$)和接近末尾的两个阶段($2\rho NC\omega\tau$ 乘 36%)外重绕时间几乎重叠。我们也可以从 β 减去 0.18，因为前半个阶段为初始重绕所重叠。

例 5　具重绕重叠的级联合并　在这种情况下对于 $T=5$ 我们使用表 5.4.3-1，得到 $\alpha\approx0.897,\beta\approx0.800-2$。几乎所有的非重叠的重绕都恰出现在初始分布之后和每两路合并之后。在一个完全的初始分布之后，最长的磁带包含数据的大约 1/g，其中 g 是“增长率”。在每个两路合并之后，在 6 条磁带的情况下重绕的数量为 d_kd_{n-k}(参见习题 5.4.3-5)，因此可以证明，在 T 条磁带的情况下，在两路合并之后的重绕量近似于该文件的

$$(2/(2T-1))(1-\cos(4\pi/(2T-1)))$$

在我们的情况下，$T=5$，这是该文件的 $\frac{2}{9}(1-\cos80^\circ)\approx0.184$，而且它出现的次数是 $0.946\ln S+0.796-2$。

例 6　向后读平衡合并　除大多数的重绕已被消去外，它和例 1 类似。从向前变成为向后这一方向变化，引起某些延迟，但是这些延迟并不重要。在最后扫描之前，将有 50-50 机会需要重绕，所以我们可以取 $\pi'=1/(2P)$。

例 7　向后读多阶段合并　由于此情况下的替代选择产生大约每 P 次改变一次方向的路段，我们必须以 S 的另一个公式来替换(3)。由习题 5.4.1-24 所提示的、一个相当好的近似是 $S=\lceil N(3+1/P)/(6P')\rceil+1$。消去了所有的重绕时间，因而表 5.4.2-1 给出 $\alpha\approx0.863,\beta\approx0.921-2$。

例 8　向后读级联合并　由表 5.4.3-1 我们有 $\alpha\approx0.897,\beta\approx0.800-2$。重绕时间可以估计为在该表中的“有拷贝的扫描”减去“无拷贝的扫描”后所得差额的两倍，假如在最后的合并之前必须重绕，以得到递增的次序，则加上 1/(2P)。

例 9　向后读振荡排序　在这种情况下，替代选择要被启动和停止许多次；每次分布 $P-1$ 到 $2P-1$ 个路段，长度平均为 P；因此，路段的平均长度近似于 $P'(2P-4/3)/P$，故我们可以估计 $S=\lceil N/((2-4/(3P))P')\rceil+1$。用一点时间来做从合并变为分布和从分布变为合并的转换；这近似于从输入磁带读入 P'个记录的时间，即 $P'C\omega_i\tau$，且它大约出现 S/P 次。重绕时间和合并时间可以像例 6 中那样估计。

例 10　向前读振荡排序　此方法不易于分析，因为在输入已穷尽后最后执行的“清除”阶段不像先前的一些阶段那样有效。若忽略这个麻烦的方面，只简单地要求做额外一次扫描，我们就可以通过置 $\alpha=1/\ln P,\beta=0$，及 $\pi'=\pi/P$ 来估计合并时间。在这种情况下路段的分布有些不同，因为未用替代选择；我们置 $P'=M/C$ 及 S

$=\lceil N/P' \rceil$。注意，通过在开销中增加大约为 $(M+2B)/M$ 的一个额外因子，有可能在分布期间使计算和读写重叠起来。在此情况下并不需要例 9 中提到的"方式转换"时间，因为它已为重绕所重叠。所以在此情况下估计的排序时间为(9)加上 $2BNC\omega_i\tau/\dot{M}$。

表 1　排序的时间估计一览

示例	P	B	P'	S'	ω	α	β	α'	β'	(9)	加到(9)中	估计数总和	实际数总和
1	3	12500	650	79	1.062	0.910	−1.000	0.303	0.000	1064		1064	1076
2	5	8300	734	70	1.094	0.795	−1.136	0.795	−1.136	1010	$\rho NC\omega_i\tau+\delta$	1113	1103
3	5	8300	734	70	1.094	0.773	−1.192	0.773	−1.192	972	$\rho NC\omega_i\tau+\delta$	1075	1127
4	4	10000	700	73	1.078	0.752	−0.944	0.000	0.720	844	$\rho NC\omega_i\tau+\delta$	947	966
5	4	10000	700	73	1.078	0.897	−1.200	0.173	0.129	972		972	992
6	3	12500	650	79	1.062	0.910	−1.000	0.000	0.167	981		981	980
7	4	10000	700	79	1.078	0.863	−1.079	0.000	0.000	922		922	907
8	4	10000	700	73	1.078	0.897	−1.200	0.098	0.117	952		952	949
9	4	10000	700	87	1.078	0.721	−1.000	0.000	0.125	846	$P'SC\omega_i\tau/P$	874	928
10	4	10000	−	100	1.078	0.721	0.000	0.180	0.000	1095	$2BNC\omega_i\tau/M$	1131	1158

表 1 表明，在这些示例中这些估计并不太坏，尽管在一些情况下，有 50s 左右的差异。例 2 和例 3 中的公式指出 6 条磁带时级联合并比多阶段还要好些，然而实际上却是多阶段更好。原因是像图 85(它示出 5 条磁带的情况)这样的图对于多阶段算法来说更接近于直线；如果有 $14\leqslant S\leqslant 15$ 和 $43\leqslant S\leqslant 55$，即近乎"完全"的级联数 15 和 55，则在 6 条磁带上级联比多阶段更好，但算法 5.4.2D 的多阶段分布，对于所有其它的 $S\leqslant 100$ 同样好或更好。当 $S\rightarrow\infty$ 时级联将优于多阶段，但 S 并不真正地趋于∞。例 9 中的估计不足也是由于类似的情况所致；多阶段优于振荡，尽管渐近理论告诉我们，对于很大的 S，振荡将是更好的。

杂记　现在宜于对磁带的合并来做一些随机的观察：

·上式公式表明，磁带排序的费用实际上是 N 乘 C 的函数，而不是独立的 N 和 C 的函数。除了一些较小的考虑(例如把 B 取作 C 的一个倍数)外，我们的公式指出，对每个含 10 个字符的 100 万个记录排序，和对每个含 100 个字符的 10 万个记录排序，所花时间大体相同。虽然在公式中未予说明，但实际上却可能有差别，这差别是由于在替代选择期间为链接诸字段所用的空间造成的。无论如何，键码的大小很难造成任何差别，除非键码是那么长和复杂，以致内部计算已不能跟上磁带了。

对于长的记录和短的键码，我们经常会想要先"分离出"这些键码，并对它们排序，而后再设法重新整理这些记录。但这个思想实际上并不行之有效，它仅仅是延迟了你的烦恼而已，因为最后的重整理过程大约要花费和通常的合并排序相同的时间。

·在编写一个有待重复使用的排序例程时，非常谨慎地来估计它的运行时间，并且把理论同实际观察到的性能进行比较，是明智的。由于排序理论已经相当成熟，

因此也就知道这个过程会显露出现存系统上输入/输出硬件或软件的缺陷；但服务工作慢于它应该有的速度，而且直到排序例程运行得实在太慢时，方才引起人的注意！

·我们对替代选择的分析是对“随机”文件进行的，但在实践中真正出现的文件经常已有一定的次序(事实上，有时人们确实在对已经排好序的文件排序，而这仅仅是为了确认)。因此经验证明，替代选择比其它类型的内部排序更可取，这比我们公式中所指出的还要突出。在向后读多阶段排序的情况下，这个优点不太明显，因为必须产生一些递减的路段；确实，R.L.Gilstad(他首先发表了多阶段合并)原来即由于这个原因而否定了向后读的技术。但他后来注意到，交替的方向仍将挑出长的递增路段。而且向后读多阶段是仅有的既适用于递减输入文件，也适用于递增输入文件的标准技术。

·替代选择的另一个优点是它允许同时进行读、写和计算。如果我们仅仅以一种显然的方式进行内部排序——填满内存，对它进行排序，然后当开始装入下一个负载时写出它——则分布扫描将大约花费两倍之长的时间！

我们已经讨论过的，唯一适合于同时读、写和计算的其它内部排序，就是堆排序。为了方便起见，假设内部的内存能存 1000 个记录，磁带上每个块能存 100 个记录。图表 A 的例 10 就是以下列策略编制的，令 $B_1B_2\cdots B_{10}$表示分成 10 个块，每个块各 100 个记录的内存的内容：

步骤 0　填满内存，并使 $B_2\cdots B_{10}$的元素满足堆(在根处具有最小元素)的诸不等式。

步骤 1　使 $B_1\cdots B_{10}$成为一个堆，然后选择出最小的 100 个记录，并且把它们移到 B_{10}。

步骤 2　写出 B_{10}，同时选择 $B_1\cdots B_9$ 的最小的 100 个记录，并把它们移到 B_9。

步骤 3　读入 B_{10}，并写出 B_9，同时选择 $B_1\cdots B_8$ 的最小的 100 个记录，并把它们移到 B_8。

……

步骤 9　读入 B_4，并写出 B_3，同时选择 B_1B_2 的最小的 100 个记录，并把它们移到 B_2，同时使堆的诸不等式在 $B_5\cdots B_{10}$中成立。

步骤 10　读入 B_3，并写出 B_2，同时对 B_1 排序并使堆的诸不等式在 $B_4\cdots B_{10}$中成立。

步骤 11　读入 B_2，并写出 B_1，同时使堆的诸不等式在 $B_3\cdots B_{10}$中成立。

步骤 12　读入 B_1，同时使堆的诸不等式在 $B_2\cdots B_{10}$中成立，返回步骤 1。 ▌

·我们已经假定要排序的记录个数 N 事先是未知的。实际上在大多数计算机的应用中，对全部文件中的记录个数保持跟踪是可能的，并且可以假定计算机系统有能力告诉我们 N 的值。这将有多大的帮助呢？可惜，不很大！我们已经看到，替代选择是非常有利的，但是它导致了一个不可预测的初始路段个数。在一个平衡合

并中,我们可以使用关于 N 的信息,以这样的方式来设置缓冲区的大小 B,即使得 S 大概将恰恰小于 P 的一个乘方;而在具有最优的虚拟路段设置的多阶段分布中,我们可以使用关于 N 的信息,来判断使用什么级别最好(参考表 5.4.2-2)。

·磁带驱动器倾向于被看作是计算机最不可靠的部分,因此在确保整个排序已经令人满意地完成之前,决不要损坏原来的输入磁带。在图表 A 的一些示例中,虽然"操作员的卸磁带时间"是令人厌烦的,但是,鉴于在一个长排序期间经常可能出错,所以重写输入还是太冒险了。

·当从向前写变成向后读时,只要不把最后的缓冲负载写到磁带上,我们可以节省一些时间;因为它总是要读进来的!图表 A 表明,这个技巧实际上节省的时间比较少,除非是振荡排序,它经常要变换方向。

·尽管一个大型的计算机系统可能有大量的磁带机,但是我们不把它们全用光可能更好些。当 P 很大时,$\log_P S$ 和 $\log_{P+1} S$ 之间的百分比差不是非常大的,而且更高阶的合并通常意味着更小的块(另外还要考虑安装所有这些空白磁带的可怜的计算机操作员)。另一方面,习题 12 描述了利用附加磁带机的一种有趣方式,即把它们组合在一起以便重叠输入/输出时间,而不必增加合并的阶。

·像 MIX 这样的机器上,都有固定的很小的块大小,在合并时几乎不需要使用任何内存。这样振荡排序变得更有吸引力,因为在合并时维持替代选择树于内存中成为可能。事实上,在这种情况下,我们可以(如 Colin J. Bell 在 1962 年提议的)改进振荡排序,每次从工作磁带合并时,同时合并一个新的初始路段到输出中。

·我们已经观察到,对多卷文件,每次只应该排序一卷,以避免过多的磁带处理。这有时称为"卷时间"应用程序。实际上,如果比较细心地编写程序,则在 6 条磁带上的一个平衡排序,能够同时对 3 个满卷排序,直到最后的合并时间为止。

为了对大量的已经各自排好序的磁带卷进行排序,极小路径长度的合并树将是最快的(参见 5.4.4 小节)。这个构造首先是由 E. H. Friend 做出的[*JACM* **3** (1956),166~167]。而后 W. H. Burge[*Information and Control* **1**(1958),181~197]指出,如果我们忽略磁带的处理时间,则合并给定(可能不等)长度路段的一种最优方式,可通过构造用路段长度做为权,具有极小加权路径长度的一株树来得到(参见 2.3.4.5 小节和 5.4.9 小节)。

·我们的讨论已经冒失地假定我们有对磁带机输入/输出指令的直接控制,而且没有复杂的系统接口来阻止我们像磁带的设计者所希望的那样有效地使用磁带。这些理想的假定,使我们洞察了磁带合并的问题,而且还可以使我们对于操作系统界面的正确设计有所了解,但应认识到,多程序设计和多进程可使情况更为复杂。

·这一节所研究的内容,是由 E. H. Friend[*JACM* **3**(1956),134~168],W. Zoberbier[*Elektronische Datenverarbeitung* **5**(1960),28~44]以及 M. A. Goetz[*Digital Computer User's Handbook* (New York:McGraw-Hill,1967),1.292~1.320]首先在出版的著作中讨论的。

小结 我们可以以如下方式,总结在磁带排序的各个不同方法的相对效率中所

学到的东西。

定理 A　*判断一个给定情况下哪个合并型式最好是困难的。* ▌

图表 A 中的示例说明，100 000 个随机排序的 100 个字符的记录(或 1 百万个 10 个字符的记录)，在现实的假定之下，可以如何利用 6 条磁带来进行排序。这么多数据填满大约一半的磁带，而且它们能在大约 15min～19min 内在 MIXT 磁带上被排序；然而可利用的磁带机有相当大的差异，而且这样一个作业的运行时间，在不同的机器上，可以小到 4min，大到 2h。在我们的例子中，路段的初始分布和内部排序使用了大约 3min 时间；最后的合并和重绕输出磁带使用大约 $4\frac{1}{2}$min；而合并的中间阶段花费的大约 $7\frac{1}{2}$min～$11\frac{1}{2}$min。

给定不能向后读的 6 条磁带，在我们的假定之下，最好的排序方法是“分磁带的多级合并”(例 4)；而对于允许向后读的磁带，最好的方法为具有一个复杂的虚拟路段设置的向后读多阶段法(例 7)。振荡排序(例 9)仅次于它。在这两种情况下级联合并提供了一个更简单的方案，它仅仅稍微慢些(例 5 和例 8)。在向前读的情况下，一个直截了当的平衡合并(例 1)惊人地有效，其部分是因为这个特定例子恰好适合于它，部分是因为它花费较小的重绕时间。

如果我们有不同数目的磁带可供利用，则情况将稍有不同。

排序生成器　由于数据以及设备的特性是千变万化的，所以几乎不可能写一个单一的外部排序程序，使它在非常多的应用中都令人满意。而且也颇难编写能真正有效处理磁带的一个程序。因此排序软件的编制是一项特别有挑战性的工作。一个排序生成器是这样一个程序，它根据描述数据格式和硬件配置的参数，产生出适合于某类特定的排序应用的机器代码。这样一个程序通常都依赖于诸如 COBOL 或 PL/I 这样的高级语言。

一个排序生成器通常提供的特性之一，是插入用户“自己的代码”的能力，这是有待加入到排序例程的第一遍和最后一遍扫描的特殊指令序列。第一遍扫描的自己的代码常常用来编辑输入记录，通常把它们缩小或扩展成为易于排序的形式。例如，假设输入记录按一个 9 字符的键码来进行排序，该键码以月-日-年的格式表示日期：

JUL041776　OCT311517　NOV051605　JUL141789　NOV071917

在第一次扫描时，3 个字母的月代码可在一份表中查出，而且月代码可以用最高有效字段在左边的数来代替：

17760704　15171031　16051105　17890714　19171107

这就减小了记录的长度并且使随后的比较简单得多(甚至还可以换成一个更紧凑的代码)。最后扫描的自己的代码，则可用来恢复原来的格式，和/或对文件进行其它想要的变动，和/或计算输出记录的某个函数。我们已经研究过的合并算法，是以这种方式组织的，即易于把最后的扫描同其它合并阶段区别开来。注意，当出现自己

的代码时，必须至少对文件进行两次扫描，即便它开始时已经是有序的亦然。改变记录大小的自己的代码，可使振荡排序难以来重叠它的某些输入/输出操作。

排序生成器也照顾到诸如磁带标识的约定等系统细节，而且它们通常提供“散列总和”或其它校验以确保不丢失或改变数据。有时还有在适当的位置停止排序及过后恢复执行的规定。最优秀的生成器允许记录有动态变化的长度[参见 D. J. Waks, *CACM* **6**(1963), 267～272]。

* **使用较少缓冲区的合并** 我们已经看到，在 P 路合并期间，为跟上磁带的急速移动，$2P+2$ 个缓冲区已足够了。现在在结束本节之前，让我们从数学上分析一下，少于$2P+2$个缓冲区时的合并时间。

两个输出缓冲区显然是可取的，因为我们可以在从一个中写出的同时，在另一个中形成下一个输出块。因此我们完全可以忽略输出的问题，而仅仅专注于输入。

假设有 $P+Q$ 个输入缓冲区，其中 $1\leqslant Q\leqslant P$。如同 L. J. Woodrum 所提议的，我们将使用这种状态的下列近似模型[*IBM Systems J.* **9**(1970), 118～144]；每读磁带上一个块需用一个时间单位。在这段时间里，没有输入缓冲区变成空的概率为 p_0，有一个变成空的概率为 p_1，有两个或更多变成空的概率为 $p_{\geqslant 2}$，等等，当完成一条磁带的读入时，我们处于 $Q+1$ 个状态之一：

状态 0 Q 个缓冲区为空；我们使用在此小节中早些时候说明的预报技术，开始从适当的文件读一个块进入 Q 个缓冲区之一。在一个时间单位之后，我们以概率 p_0 转到状态 1，否则我们留在状态 0。

状态 1 $Q-1$ 个缓冲区为空；预报适当的文件，我们开始读入 $Q-1$ 个缓冲区之一。在一个时间单位之后，我们以概率 p_0 转到状态 2，以概率 p_1 转入状态 1，以概率 $p_{\geqslant 2}$转入状态 0。

⋮

状态 $Q-1$ 一个缓冲区为空；预报适当的文件，我们开始读入此缓冲区。在一个时间单位之后，我们以概率 p_0 转到状态 Q，以概率 p_1 转到状态 $Q-1$，以概率 p_{Q-1}转到状态 1 和以概率 $p_{\geqslant Q}$转到状态 0。

状态 Q 所有缓冲区都被填满。读磁带停止平均 μ 个时间单位，而后转到状态 $Q-1$。

我们以状态 0 开始。这个状态的模型对应于一个马尔可夫过程(参见习题 2.3.4.2-26)，它可以以下列有趣的方式通过生成函数来进行分析：设 z 是一个任意参数，而且假定，每当我们有读磁带机会时，决定读磁带的概率为 z，而决定结束算法的概率为 $1-z$。现在令 $g_Q(z)=\Sigma_{n\geqslant 0}a_n^{(Q)}z^n(1-z)$是在这样一个过程中出现状态 Q 的平均次数；由此得出 $a_n^{(Q)}$ 是当恰好读了 n 个块时状态 Q 出现的平均次数。于是 $n+a_n^{(Q)}\mu$ 是输入加上计算的平均总时间。如果如同在$(2P+2)$个缓冲区的算法中那样，我们有完全的重叠，则总共的时间将仅仅是 n 个单位，所以 $a_n^{(Q)}\mu$ 表示

"读挂起"的时间。

对于 $0\leqslant i,j\leqslant Q+1$,其中 $Q+1$ 是一个新的"被停止"状态。令 A_{ij} 是在此过程中我们从状态 i 转到状态 j 的概率。例如,对于小的 Q,A 矩阵取如下的形式:

$$Q=1:\begin{pmatrix} p_{\geqslant 1}z & p_0z & 1-z \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$Q=2:\begin{pmatrix} p_{\geqslant 1}z & p_0z & 0 & 1-z \\ p_{\geqslant 2}z & p_1z & p_0z & 1-z \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$Q=3:\begin{pmatrix} p_{\geqslant 1}z & p_0z & 0 & 0 & 1-z \\ p_{\geqslant 2}z & p_1z & p_0z & 0 & 1-z \\ p_{\geqslant 3}z & p_2z & p_1z & p_0z & 1-z \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

习题 2.3.4.2-26(b)告诉我们,$g_Q(z)=$余因式$_{Q0}(I-A)/\det(I-A)$。于是,例如当 $Q=1$ 时,我们有

$$g_1(z)=\det\begin{pmatrix} 0 & -p_0z & z-1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}\Big/\det\begin{pmatrix} 1-p_{\geqslant 1}z & -p_0z & z-1 \\ -1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}=$$

$$\frac{p_0z}{1-p_1z-p_0z}=\frac{p_0z}{1-z}=\sum_{n\geqslant 0}np_0z^n(1-z)$$

所以 $a_n^{(1)}=np_0$,这当然是预先就显然的,因为当 $Q=1$ 时问题是很简单的。当 $Q=2$ 时(见习题 14)类似的计算给出不大显然的公式

$$a_n^{(2)}=\frac{p_0^2n}{1-p_1}-\frac{p_0^2(1-p_1^n)}{(1-p_1)^2} \tag{10}$$

一般地说,我们可以证明,当 $n\to\infty$ 时 $a_n^{(Q)}$ 的形式为 $\alpha^{(Q)}n+O(1)$,其中常数 $\alpha^{(Q)}$ 不是太难计算的(见习题 15)。结果为 $\alpha^{(3)}=p_0^3/((1-p_1)^2-p_0p_2)$。

鉴于合并的本性,我们有相当的理由来确定 $\mu=1/P$ 以及一个二项式分布

$$p_k=\binom{P}{k}\left(\frac{1}{P}\right)^k\left(\frac{P-1}{P}\right)^{P-k}$$

例如,当 $P=5$ 时,我们有 $p_0=.32768$,$p_1=.4096$,$p_2=.2048$,$p_3=.0512$,$p_4=.0064$,以及 $p_5=.00032$,因此 $\alpha^{(1)}\approx 0.328$,$\alpha^{(2)}\approx 0.182$,以及 $\alpha^{(3)}\approx 0.125$。换句话说,如果我们使用 5+3 个输入缓冲区代替 5+5 个输入缓冲区,则可以预期一个额外的大约 0.125/5≈2.5%的"读挂起"时间。

当然,这个模型仅仅是一个非常粗略的近似,我们知道,当 $Q=P$ 时,全然没有挂起时间,但这个模型说有。对于较小的 Q,额外的读挂起时间大约恰恰抵消通过较大的块所节省的开销,所以简单的 $Q=P$ 的方案似乎是合理的。

习 题

1.[*13*] 假设在没有块间的间隔时,每条磁带正好包含 23 000 000 个字符。求当磁带上的每个块包含 n 个字符时,每条磁带的准确字符数的一个公式。

2.[*15*] 说明为什么图 84 的第 6 行中文件 2 的第一个缓冲区完全是空的?

3.[*20*] 如果仅有 $2P-1$ 个输入缓冲区而不是 $2P$ 个,则算法 F 能正确工作吗?若能,则证明之;若不能,则给出一个使它不对的例子。

4.[*20*] 如何改变算法 F,使得当 $P=1$ 时它也能够工作?

▶5.[*21*] 当相等键码出现在不同的文件中时,在预报过程中就必须非常小心。说明为什么,并通过更精确地定义算法 F 的合并和预报操作来说明怎样避免这个问题?

6.[*22*] 为了把算法 5.4.3C 转换成对于 $T+1$ 条磁带的具有重绕重叠的级联合并的一个算法,对算法 5.4.3C 应该做些什么改动?

▶7.[*26*] 图表 A 例 7 中的初始分布产生

$$(A_1D_1)^{11}\quad D_1(A_1D_1)^{10}\quad D_1(A_1D_1)^9\quad D_1(A_1D_1)^7$$

于磁带 1~4 上,这里 $(A_1D_1)^7$ 指的是 $A_1D_1A_1D_1A_1D_1A_1D_1A_1D_1A_1D_1A_1D_1$。说明如何以"最好"的方式来插入诸 A_0 和 D_0(所谓最好,指的是在合并时所处理的初始路段总数为极小),使分布变成为

$$A(DA)^{14}\quad (DA)^{28}\quad (DA)^{26}\quad (DA)^{22}$$

提示:为了保持奇偶性,有必要把许多 A_0 和 D_0 作为相邻的对偶插入。每个初始路段的合并数,可以像在习题 5.4.4-5 中那样来计算;由于相邻的路段总有相邻的合并数,就会出现某些简化。

8.[*20*] 图表 A 说明,路段初始分布的大多数方案(对于级联合并的初始分布例外),都趋向于把相继的路段置到不同的磁带上。如果相继的路段放到相同的磁带上,则我们可以节省停止/启动时间;如果因此而修改分布算法,以减少磁带的转换次数,是一个好的想法吗?

▶9.[*22*] 如果我们对于排序都用 $T=6$ 条磁带,而不是像在例 7 中那样用 $T=5$ 条磁带,试估计图表 A 中的向后读多阶段算法将会多长。避免使用输入磁带是明智的吗?

10.[*M23*] 使用 5.4.2 小节和 5.4.3 小节的分析,证明在一个标准的 6 条磁带的多阶段或级联合并期间,每次重绕的长度很少超过文件的大约 54%(初始和最后的重绕除外,它们遍及整个文件)。

11.[*23*] 通过修改表 1 中适当的项,试估计,如果我们有一个联合的低速/高速重绕,则图表 A 的前 9 个例子将花费多长时间?假定磁带未装满 1/4 时 $\rho=1$,并假定更满的磁带的重绕时间近似于 5s 加上当 $\rho=\frac{1}{5}$ 时花费的时间。试改变例 8,使得它使用有拷贝的级联合并,因为在这种情况下重绕和向前读比拷贝更慢[提示:使用习题 10 的结果]。

12.[*40*] 考虑把 6 条磁带划分成 3 对磁带,每对在 $T=3$ 的多阶段合并中起着一条磁带的作用。每对的一条磁带将包含块 1,3,5…而另一条磁带将包含块 2,4,6,…;这样一来,合并时,实质上我们总有两条输入磁带和两条输出磁带在活动,这等效于加倍了合并的速度。

a)试找出一个适当的方式把算法 F 推广到这一情况。应有多少个缓冲区？

b)如果用这个方法对 100 000 个 100 个字符的记录排序，考虑向前读和向后读两种情况，试估计总共的运行时间将是多少？

13.[*20*] 按照算法 5.4.5B 中的定义，一个 5 磁带振荡排序可否用来对 4 个满卷的输入数据排序，直到最后合并时为止？

14.[*M19*] 推导(10)。

15.[*HM29*] 证明 $g_Q(z)=h_Q(z)/(1-z)$，其中 $h_Q(z)$ 是单位圆里边没有奇点的 z 的一个有理函数；因此当 $n\to\infty$ 时，$a_n^{(Q)}=h_Q(1)n+O(1)$，特别证明

$$h_3(1)=\det\begin{pmatrix}0 & -p_0 & 0 & 0\\ 0 & 1-p_1 & -p_0 & 0\\ 0 & -p_2 & 1-p_1 & -p_0\\ 1 & 0 & 0 & 0\end{pmatrix}\Bigg/\det\begin{pmatrix}1 & -p_0 & 0 & 0\\ 1 & 1-p_1 & -p_0 & 0\\ 1 & -p_2 & 1-p_1 & -p_0\\ 0 & 0 & -1 & 1\end{pmatrix}$$

16.[*41*] 假定可利用 3、4 或 5 条磁带时，像在图表 A 中那样画出图表，详细研究对 100 000 个 100 字符的记录的排序问题。

*5.4.7　外部基数排序

前面几节已经讨论了通过合并进行磁带排序的过程；但是还有另外一种用磁带进行排序的方式，它的依据是机械的卡片排序机(参见 5.2.5 小节)所用的基数排序原理。这个方法有时称为分布排序、列排序、袋排序、数字排序、分开排序等；可以证明，它实际上恰与合并相对立！

例如，假设有 4 条磁带和仅有 8 种可能的键码：0,1,2,3,4,5,6,7。如果输入数据在磁带 1 上，则开始我们可以把所有的偶键码传送到 T3，把所有的奇键码送到 T4：

	T1	T2	T3	T4
给定	{0,1,2,3,4,5,6,7}	–	–	–
扫描 1	–	–	{0,2,4,6}	{1,3,5,7}

现在我们重绕，并读 T3 而后读 T4，把{0,1,4,5}置于 T1 上，{2,3,6,7}置于 T2 上：

	T1	T2	T3	T4
扫描 2	{0,4}{1,5}	{2,6}{3,7}	–	–

符号“{0,4}{1,5}”表示一个文件，这个文件包含某些记录，它们的键码都是 0 或 4，后边跟着键码都是 1 或 5 的记录。注意，T1 现在包含的诸键码的中间二进数字均为 0。在再次重绕和分布 0,1,2,3 到 T3 以及 4,5,6,7 到 T4 之后，有

	T1	T2	T3	T4
扫描 3			{0}{1}{2}{3}	{4}{5}{6}{7}

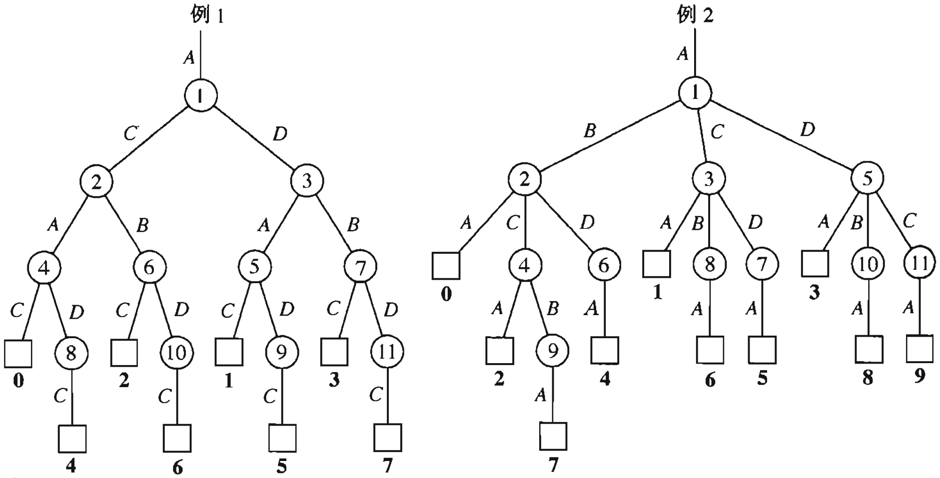
现在我们可以通过拷贝 T4 到 T3 的末尾来结束。一般地，如果键码的范围是从 0 到 2^k-1，则我们可以利用 k 趟扫描，以类似的方式对文件排序，后边接之以最后的“收集”阶段，它从一条磁带拷贝大约一半的数据到另一条磁带。若用 6 条磁带，则可以以类似的方式，使用基数 3 的表示，在 k 趟扫描中对从 0 到 3^k-1 的键码排序。

也可以使用部分扫描的方法。例如,假设有 10 种可能的键码{0,1,…9},考虑由 R. L. Ashenhurst 给出的下列过程[*Theory of Switching*, Progress Report BL-7 (Harvard Univ. Comp. Laboratory: May 1954), I.1~I.76]:

阶段	T1	T2	T3	T4	扫描
	{0,1,…,9}	–	–	–	
1	–	{0,2,4,7}	{1,5,6}	{3,8,9}	1.0
2	{0}	–	{1,5,6}{2,7}	{3,8,9}{4}	0.4
3	{0}{1}{2}	{6}{7}	–	{3,8,9}{4}{5}	0.5
4	{0}{1}{2}{3}	{6}{7}{8}	{9}	{4}{5]	0.3
收集	{0}{1}{2}{3}{4}…{9}				0.6
					2.8

如果每个键码值出现的机会大约为 1/10,则上述过程只花费 2.8 趟扫描来对 10 个键码排序,而第一个例子则需要 3.5 次扫描来对仅仅 8 个键码排序。因此我们发现,无论是基数排序还是合并,由于所使用的分布型式巧妙不同,其结果可以是很不一样的。

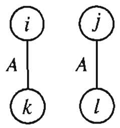
上述例子中的分布型式可以方便地表示成树结构:



这些树圆形的内部节点编号为 1,2,3,…,对应于这一过程的步骤 1,2,3,…。所有树的旁边分别标以磁带名 A,B,C,D(代替 T1,T2,T3,T4),以说明这些记录的去处。正方形的外部节点表示一个文件仅包含有一个键码的部分,该键码以黑体示于此节点之下。在正方形节点上方的线旁为输出磁带的名字(在第一个例子中为 C,在第二个例子中为 A)。

于是,例 1 中步骤 3 的组成为:从磁带 D 读入并写 1 和 5 到磁带 A 上,写 3 和 7 到磁带 B 上。如果我们假定每个键码都同样经常地出现,则不难看出,所执行的扫描趟数等于树的外部路径长度除以外部节点数。

由于磁带的顺序性,以及向前读的 - fifo 的原则,我们不能简单地使用任何有标号的树来作为一个分布型式的基础。在例 1 的树中,在步骤 2 和步骤 3 期间数据写到磁带 A 上;在我们使用步骤 3 期间所写的数据之前,必须首先使用步骤 2 期间所写的数据。一般地说,如果在步骤 i 和 j 期间写到一个磁带上,其中 $i<j$,则我们必须首先使用在步骤 i 期间写的数据;当树包含形如



$i<j$

的两个分支时,必须有 $k<l$。并且不能在步骤 k 和 l 之间向磁带 A 上写任何东西,因为我们必须在读和写之间进行重绕。

已经做了 5.4.4 小节习题的读者,现在会立即觉察到,可以表示在 T 条磁带上向前读基数排序的树,恰恰就是"强 T-fifo"树,它表征了 T 条磁带上的向前读的合并排序(见习题 5.4.4-20)! 惟一的差别是,我们在这里考虑的树的所有外部节点都有相同的磁带标号。只要假定一个最后的"收集阶段",这个阶段把所有记录传送到一条输出磁带上,我们就可以去掉这个限制;或者也可以附加这个限制到 T-fifo 树的诸规则上,为此要求合并排序的初始分布扫描,能明显地在对应的合并树中表示出来。

换句话说,*每一个合并型式对应于一个分布型式,而每一个分布型式对应于一个合并型式*。稍经考虑后就知道为什么这样:我们按反方向做合并排序,首先"拆开"最后的输出成为一些子文件,这些子文件又拆开成其它的,等等。最后,我们将把这个文件拆开成为 S 个路段。实现这样一个型式是可能的充要条件是,对应的基数排序分布型式对于 S 个键码来说是可能的。合并与分布之间的这种对偶性几乎是完全的;它仅在一个方面不成立,即输入磁带的内容必须在不同时间里保存。

本节开始时所讨论的 8 个键码的例子,显然对偶于 4 条磁带上的一个平衡合并。部分扫描的 10 个键码的例子,对应于下列 10 路段的合并型式(如果去掉拷贝阶段——即树中的步骤 6～11——的话):

	T1	T2	T3	T4
初始分布	1^4	1^3	1^1	1^2
树步骤 5	1^3	1^2	–	$1^2 3^1$
树步骤 4	1^2	1^1	2^1	$1^2 3^1$
树步骤 3	1^1	–	$2^1 3^1$	$1^1 3^1$
树步骤 2	–	4^1	3^1	3^1
树步骤 1	10^1	–	–	–

如果把此同基数排序进行比较,则我们看到,这些方法实际上都有相同的构造,但在时间上是颠倒的,同时磁带的内容也从后面颠倒到前头:$1^2 3^1$(两个长度皆为 1 的路段,后边接有长度为 3 的一个路段)对应于 {3,8,9}{4}{5}(两个子文件,每个包含 1 个键码,前面还有包含 3 个键码的一个子文件)。

换一种方式，我们原则上也可以构造一个基数排序，使其对偶于多阶段合并，另一个对偶于级联合并，等等。例如，在 5.4.2 节开始所说明的在 3 条磁带上的 21 个路段的多阶段合并，对应于下列有趣的基数排序：

阶段	T1	T2	T3
	{0,1,…,20}	–	–
1	–	{0,2,4,5,7,9,10,12,13,15,17,18,20}	{1,3,6,8,11,14,16,19}
2	{0,5,10,13,18}	–	{1,3,6,8,11,14,16,19} {2,4,7,9,12,15,17,20}
3	{0,5,10,13,18}{1,6,11,14,19} {2,7,12,15,20}	{3,8,16}{4,9,17}	–
4	–	{3,8,16}{4,9,17}{5,10,18} {6,11,19}{7,12,20}	{0,13}{1,14}{2,15}
5	{8}{9}{10}{11}{12}	–	{0,13}{1,14}{2,15} {3,16}…{7,20}
6	{8}{9}{10}{11}{12}{13}…{20}	{0}{1}…{7}	–

这里用来决定在每个步骤中哪些键码进到哪些磁带上的分布规则，似乎是玄妙的，但是事实上它与斐波那契数系统有着一种简单的联系(见习题 2)!

向后读 基数排序和合并之间的对偶性也可应用于向后读的算法。我们已经在 5.4.4 小节定义了"T-lifo 树"，容易看出，它们对应于基数排序，也对应于合并排序。

实际上，在 1946 年 John Mauchly 就已考虑过向后读的基数排序，那是关于排序的最早发表的论文之一(参见 5.5 节)；Mauchly 实际上给出了下列构造：

阶段	T1	T2	T3	T4
	–	{0,1,2,…9}	–	–
1	{4,5}	–	{2,3,6,7}	{0,1,8,9}
2	{4,5}{2,7}	{3,6}	–	{0,1,8,9}
3	{4,5}{2,7}{0,9}	{3,6}{1,8}	–	–
4	{4,5}{2,7}	{3,6}{1,8}	{9}	{0}
……	……	……	……	……
8	–	–	{9}{8}{7}{6}{5}	{0}{1}{2}{3}{4}
C	–	–	–	{0}{1}{2}{3}{4}{5}…{9}

他的方案并不是最有效的，但它却是有趣的，因为它说明，在 1946 年即已考虑过在基数排序中使用部分扫描方法，尽管直到大约 1960 年以前它们还未出现在关于合并的著作中。

向后读分布型式的一个有效的构造已经由 A. Bayes 提出[*CACM* **11**(1968)，491～493]；给定 $P+1$ 条磁带和 S 个键码，把这些键码分成为 P 个子文件，每一个含 $\lfloor S/P \rfloor$ 或 $\lceil S/P \rceil$ 个键码，并递归地应用这个过程于每个子文件。当 $S<2P$ 时，一个子文件应仅由最小的键码所组成，而且它应被写到输出文件中(R. M. Karp 的一般

先序构造包括这个方法作为其特殊情况，Karp 的方法出现在 5.4.4 小节末尾）。

向后读使合并稍微复杂化了，因为它颠倒了路段的次序。对于基数排序有相应的影响：它使得结果是稳定的或“不稳定的”取决于在树中达到什么级。在一个向后读的基数排序——其中某些外部节点在奇数级上，而某些在偶数级上——之后，具有相等键码的不同记录的相对次序，对于某些键码将和原始次序相同，但对于其它键码，则将和原来的次序相反（参见习题 6）。

振荡合并排序也有它们在对偶意义下的配对物。在一个振荡基数排序中，我们继续分开键码，直到达到仅有一个键码的子文件为止，或者直到子文件小到足以在内部进行排序为止。对这样的子文件排序并把它们写到输出磁带上，然后继续分下去。例如，如果有 3 条工作磁带和一条输出磁带，并且如果键码是二进制数，则我们可以开始把形如“$0x$”的键码放在磁带 T1 上，把键码“$1x$”放到 T2 上。如果 T1 接收一个以上的内存负载，则再次扫描它，并且置“$00x$”到 T2 上和“$01x$”到 T3 上。现在如果“$00x$”子文件短到足以在内部进行排序，则排序并输出其结果，然后继续来处理“$01x$”子文件。这样一个方法被 E. H. Friend 称为“级联伪基数排序”[*JACM* **3**(1956)，157～159]；H. Nagler 进一步发展了这个方法[*JACM* **6**(1959)，459～468]，他给它以精采的名称“两头蛇排序”。而 C. H. Gaudette 也发展了这一方法[*IBM Tech. Disclosure Bull.* **12**(April，1970)，1849～1853]。

基数排序胜过合并吗？　对偶性原理的一个重要结果是基数排序通常都差于合并排序。这是由于替代选择技术使合并排序具有一定的优势所致；没有明显的方式来安排基数排序，使得我们能利用同时处理一个以上内存负载的内部排序。确实，振荡基数排序将经常产生稍微小于一个内存负载的子文件，所以分布型式将对应于一株树，其外部节点较使用合并和替代选择时出现的外部节点多得多。因此，树的外部路径长度——即排序时间——就增加了（参见习题 5.3.1-33）。

另一方面，外部基数排序有其应用。例如，假设我们有一个包含某大公司所有职员名字的文件，名字以字母顺序出现；这个公司有 10 个部门，而且希望按部门来对文件进行排序，同时在每个部门中仍保留职员的字母顺序。如果文件很长，则这就是一个应用稳定的基数排序的理想情况，因为属于每个部门的记录数大概将多于通过替代选择产生的初始路段中所得到的记录个数。一般说来，如果键码的值范围如此之小，以致具有同一键码的记录集合相当内存大小的两倍以上，则使用一项基数排序技术是明智的。

在 5.2.5 小节我们已经看到，在某些高速计算机上，内部基数排序优于合并排序，因为基数排序算法的“内循环”避免了复杂的分支。如果外存特别快，则这样的机器不可能足够快地合并数据，以赶上输入输出设备。因此，可以证明在这种情况下基数排序优于合并，特别是，如果已知键码是一致地分布时。

习　题

1.[*20*] 在 5.4 节接近开始处，我们定义了具有参数 P，$1 \leqslant P < T$ 的一般的 T 条磁带的平衡

合并。证明其对应于一个使用一个混合基数系统的基数排序。

2.[*M28*] 正文中说明了对于21个键码的3磁带多阶段基数排序。试把它推广到 F_n 个键码的情况;说明在每个阶段的最后,什么键码出现在什么磁带上[提示:考虑斐波那契数系统,习题1.2.8-34]。

3.[*M35*] 把习题2的结果推广到4条磁带或更多磁带上的多阶段基数排序(参见习题5.4.2-10)。

4.[*M23*] 试证明,Ashenhurst的分布型式是不向后读的、在4条磁带上对10个键码进行排序的最好方式,因为在所有"强4-fifo树"中,与这种型式相应的树有极小的外部路径长度(于是如果我们忽略重绕时间,则它实质上是最好的方法)。

5.[*15*] 画出对应于10个键码的Mauchly向后读基数排序的4-lifo树。

▶6.[*20*] 某个文件包含两位数字的键码00,01,…,99。在执行了对于个位数字的Mauchly基数排序之后,交换T2磁带和T4磁带的作用,我们可以对十位数字重复同一方案。问在T2磁带上键码最终将以什么次序出现?

7.[*21*] 对偶性原理也适用于多卷文件吗?

*5.4.8 双磁带排序

由于我们需要3条磁带来实现一个没有过多的磁带运动的合并过程,因此探索如何只用两条磁带就实现一个合理的外部排序是有趣的。

H.B.Demuth于1956年提出的一个方法,是综合替代选择和冒泡排序的一种排序。假定输入在磁带T1上,开始读 $P+1$ 个记录到内存中。现在输出其键码为最小的记录到磁带T2上,并且代之以下一个输入记录。维护一个选择树或 $P+1$ 个元素的优先队,可继续输出当前内存中具最小键码的记录。当最终穷尽输入时,这个文件的最大的 P 个键码将出现在内存中;以递增次序输出它们。现在重绕这两条磁带并且通过从T2读和向T1写来重复这个过程;每趟这样的扫描,都至少安排好又一组 P 个记录到它们的适当位置。程序中可加进一个简单的检验,以确定整个文件何时排好序。最多需要 $\lceil (N-1)/P \rceil$ 趟扫描。

稍经考虑后表明,这个过程的每一趟扫描,实际上就等价于冒泡排序(算法5.2.2B)的 P 趟连续扫描。如果一个元素有 P 个或更多的反序,则当它输入时,它将比树中的一切都小,所以它将立即被输出(因此失去 P 个反序)。如果一个元素有少于 P 个反序,则它将进入到选择树中,而且将在所有比它大的键码之前先被输出——由此失去它的全部反序。当 $P=1$ 时,由定理5.2.2I,这恰是冒泡排序的情况。

因此扫描的总趟数将是 $\lceil I/P \rceil$,其中 I 是各元素反序的极大个数,按5.2.2节中发展的理论,I 的平均值为 $N-\sqrt{\pi N/2}+2/3+O(1/\sqrt{N})$。

如果文件不比内存容量大很多,或者它几乎一开始就是排好序的,则这个 P 阶的冒泡排序将是相当快的;事实上,甚至当有额外的磁带机可利用时,使用它也可能是更有利的,因为空白带还必须要由一个操作员来安装。但是,两磁带冒泡排序对相当长的随机排序文件,将运行得十分慢,因为它的运行时间近似于同 N^2 成比例。

让我们来看看对于5.4.6小节的100 000个记录的例子,这个方法如何实现?

我们需要明智地选择 P,以便补偿同时进行读、写和计算时的块间间隔。由于这个例子假定每个记录的长度是 100 个字符,有 100 000 个字符将填入内存,故可以通过置

$$100(P+1)+4B=100\,000 \tag{1}$$

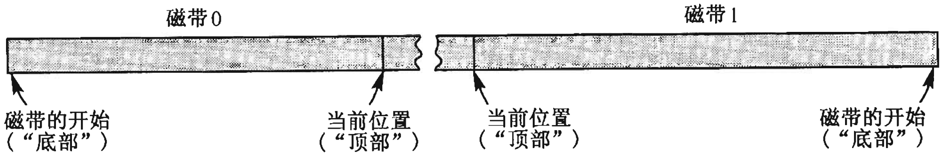
为大小为 B 的两个输入缓冲区和两个输出缓冲区提供位置。利用 5.4.6 小节的符号,每趟扫描的运行时间将大约是

$$NC\omega\tau(1+\rho), \qquad \omega=(B+\gamma)/B \tag{2}$$

由于扫描数同 P 成反比,我们需要选择 B 为 100 的倍数,它把量 ω/P 极小化。初等微积分说明,当 B 近似于 $\sqrt{24975\gamma+\gamma^2}-\gamma$ 时出现极小,所以我们取 $B=3000$, $P=879$。在上述公式中置 $N=100\,000$,就看出扫描的次数$\lceil I/P\rceil$将大约是 114,而总共的运行时间将近似于 8.57h(为方便起见,假定初始输入和最后的输出也是 $B=3000$)。这大约相当于满卷数据的 0.44 倍;一个满卷将大约有 5 倍之长。如果周期地中断算法的运行,把有最大键码的记录写到一条辅助磁带上并把它卸下,那么我们可做某些改进,因为一旦这样一些记录已按顺序放置了,所需要的只不过是向前和向后拷贝它们罢了。

快速排序的应用　另一个以近乎顺序的方式遍历数据的内部排序方法是分划交换或快速排序过程算法 5.2.2Q,我们能否使它适应两条磁带呢[N. B. Yoash, *CACM* **8**(1965),649]?

利用向后读,即不难看到这可以如何完成。假设把两条磁带编号为 0 和 1,而且想像文件的安排如下:



每一条磁带用作一个栈;把它们像这样放在一起,就有可能把文件看做一个线性表,其中,通过从一个栈拷贝到另一个栈,我们可以左右移动当前的位置。下列的递归子例程定义了一个适当的排序过程:

·SORT00 [对磁带 0 顶部的子文件排序,并且把它记回到磁带 0 上]

如果这个子文件在内存中放得下,则对它内部排序并把它返回磁带上。否则就从这个子文件选择一个记录 R,令它的键码为 K。在磁带 0 上向后读,拷贝其键码 $>K$ 的所有记录,在磁带 1 的顶部形成一个新的子文件。现在在磁带 0 上向前读,拷贝其键码 $=K$ 的所有记录到磁带 1 上。然后再次向后读,把键码 $<K$ 的所有记录拷贝到磁带 1 上。通过对于 $<K$ 的键码执行 SORT10 而完成排序,然后把 $=K$ 的键码拷贝到磁带 0 上,最后对于 $>K$ 的键码执行 SORT10。

·SORT01 [对磁带 0 顶部的子文件排序并把它写到磁带 1 上] 和 SORT00 一样,

但是最后的“SORT10”改为“SORT11”,之后紧接着拷贝$\leqslant K$ 的键码到磁带 1 上。

·SORT10［对磁带 1 顶部的子文件排序并把它写到磁带 0 上］和 SORT01 一样,只是交换 0 和 1,以及交换<和>。

·SORT11［对磁带 1 顶部的子文件排序并且把它送回到磁带 1 上］和 SORT00 一样,交换 0 和 1 以及交换<和>。

这些子例程的递归性,可以毫无困难地通过在磁带上存放适当的控制信息,来加以处理。

如果我们假定数据是按随机次序排列的,同时有相同键码的概率可予忽略,则对这个算法的运行时间可估计如下:设 M 是装满内部内存的记录个数,令 X_N 是当 $N>M$ 时在应用 SORT00 或 SORT11 于 N 个记录的一个子文件时所平均读的记录个数,令 Y_N 是对于 SORT01 或 SORT10 对应的量,于是我们有

$$X_N = \begin{cases} 0 & \text{如果 } N \leqslant M \\ 3N + 1 + \frac{1}{N}\sum_{0\leqslant k<N}(Y_k + Y_{N-1-k}) & \text{如果 } N > M \end{cases}$$
$$Y_N = \begin{cases} 0 & \text{如果 } N \leqslant M \\ 3N + 2 + \frac{1}{N}\sum_{0\leqslant k<N}(Y_k + X_{N-1-k} + k) & \text{如果 } N > M \end{cases} \tag{3}$$

这些递推式的解(见习题 2)表明,当 $N\to\infty$ 时,在外部分划阶段期间读磁带的总数量,平均说来,将是 $6\frac{2}{3}N\ln N + O(N)$。从等式 5.2.2-(25)我们还知道,内部排序阶段的平均数将是 $2(N+1)/(M+2)-1$。

如果把这一分析应用于 5.4.6 小节的 100 000 记录的例子,并且利用 25 000 个字符的缓冲区,以及假定对于 $n\leqslant M=1000$ 个记录的一个子文件,排序时间是 $2nC\omega\tau$,则我们得到近似于 103min 的平均排序时间(如同图表 A 中那样包括最后的重绕时间)。这样快速排序方法平均说来还不坏,当然,它的最坏情况证明甚至比上面讨论过的冒泡排序更为糟糕!

基数排序 基数交换方法(算法 5.2.2R)可以以类似的方式适用于两条磁带的排序,因为它很像快速排序。使这两种方法都有效的“技巧”,是不止一次地读一个文件的思想,这是我们以前的磁带算法所从未做过的。

同样的技巧亦可应用于在两条磁带上进行一种通常的“首先比较最低位有效数字”的基数排序。给定磁带 T1 上的输入数据,我们把所有其键码在二进制符号下以 0 结尾的记录都拷贝到 T2 上;然后在重绕 T1 之后,再次读它,同时拷贝其键码以 1 结尾的记录。现在两条磁带都被重绕,而且做类似的一对扫描,并且交换 T1 和 T2 的作用,并使用第二个最低位有效二进制数字。这时,T1 将包含其键码为$(\cdots 00)_2$ 的所有记录,紧接着是其键码为$(\cdots 01)_2$ 的,然后是键码为$(\cdots 10)_2$ 的,然后是$(\cdots 11)_2$的。如果键码是 b 位长,则为了完成这个排序,我们仅仅需要对文件进行 $2b$ 次扫描。

对于某些审慎地选择的数 b,这样一种基数排序可仅应用于键码前面的 b 位;如果键码是一致地分布的,这将使反序的个数减少大约 2^b 的一个因子,所以可以用较少次扫描的 P 路冒泡排序来完成这一作业。这一方法仅以向前的方向读磁带。

A. I. Nikitin 和 L. I. Sholmov[*Kibernetika* **2**,6(1966),79～84]提出了对于两条磁带分布排序的一个新颖但稍更复杂的方法。计算其前几位为各种可能组合的键码的个数,并根据这些计数构造人为的键码 $\kappa_1,\kappa_2,\cdots,\kappa_M$,使得对于每个 i,位于 κ_i 和 κ_{i+1} 之间的实际键码的个数,是在预先确定的极限 P_1 和 P_2 之间。于是,M 位于 $\lceil N/P_2\rceil$ 和 $\lceil N/P_1\rceil$ 之间。如果前几位的计数未给出充分的信息以确定这样的 $\kappa_1,\kappa_2,\cdots,\kappa_M$,就再进行一次或更多次的扫描,以便对于某些最高有效位的组合,算出各种较低有效位型式的出现频率。在构造了人为的键码 $\kappa_1,\kappa_2,\cdots,\kappa_M$ 的表之后,$2\lceil \lg M\rceil$ 次进一步的扫描足以来完成这个排序(这个方法要求同 N 成比例的存储空间,所以当 $N\to\infty$ 时它不能用于外部排序。实际上我们将不对多卷文件使用此技术,所以 M 将总是比较小的,而且人为的键码一定能从容地存入内存)。

更多磁带的模拟 F. C. Hennie 和 R. E. Stearns 想出了仅在两条磁带上模拟 k 条磁带的一般技术,其方法是,所需要的磁带运动仅以 $O(\log L)$ 的一个因子增加,其中 L 是在任一条磁带上遍历的极大距离[*JACM* **13**(1966),533～546]。他们的构造,如同 R. M. Karp 所提出的下列方法那样,在排序的情况下,可稍做简化。

利用两条磁带 T1 和 T2,我们来模拟一个通常的 4 条磁带的平衡合并。首先,T1 用来保存被模拟的磁带的内容,其方式由图 86 给出;我们想像,对于每一条被模拟的磁带,数据写到 4 个"磁道"上(事实上,磁带没有这样的"磁道",块 1,5,9,13,…被想像为磁道 1,而块 2,6,10,14,…被想像为磁道 2,等等)。另一条磁带 T2,仅用作辅助存储,以帮助 T1 移动其数据。

	区域 0	区域 1		区域 2				区域 3					
磁道 1	1	5	9	13	17	21	25	29	33	37	41	45	49
磁道 2	2	6	10	14	18	22	26	30	34	38	42	46	50
磁道 3	3	7	11	15	19	23	27	31	35	39	43	47	51
磁道 4	4	8	12	16	20	24	28	32	36	40	44	48	52

图 86 在 Hennie-Stearns 构造中磁带 1 的安排,非空白的区域磁带有阴影

每个磁道的块组成一些区域,每个区域分别包含 $1,2,4,8,\cdots,2^k,\cdots$ 个块,在每个磁道上的区域 k,或者正好装满 2^k 个数据块,或者完全是空白。例如,在图 86 中,磁道 1 在区域 1 和 3 中有数据;磁道 2 在区域 0,1,2 有数据;磁道 3 在区域 0 和 2;磁道 4 在区域 1;其余的区域都是空白。

假设我们正从磁道 1 和磁道 2 把数据合并到磁道 3。计算机的内存中有两个缓冲区用于两路合并的输入,加上第三个缓冲区用作输出。当磁道 1 的输入缓冲区变空时,我们可以按如下方式重新填满它:找出磁道 1 上第一个非空的区域,比如说,区域 k,拷贝它的第一个块到输入缓冲区;然后拷贝其它 2^k-1 个块的数据到 T2

上,并把它们移动到磁道 1 的区域 0,1,…$k-1$(区域 0,1,…,$k-1$ 现在被填满而区域 k 是空白)。每当磁道 2 的输入缓冲区变空时,一个类似的过程用来重新填满磁道 2 的输入缓冲区。当输出缓冲区准备好要写到磁道 3 时,我们颠倒这一过程,扫描整个 T1 以找出磁道 3 上的第一个空白区域,比如说区域 k,同时从区域 0,1,…,$k-1$ 拷贝数据到 T2 上。由输出缓冲区的内容加以扩充的 T2 上的数据,现在用来填满磁道 3 的区域 k。

这一过程要求有能力写到磁带 1 的中间,而不破坏该磁带上随后的信息,如同向前读振荡排序的情况(5.4.5 小节)那样。如果采用适当的预防措施,就有可能可靠地做到这一点。

为把磁道 1 的 2^l-1 个块送进内存,所需要的磁带运动的数量是 $\Sigma_{0\leqslant k<l}2^{l-1-k}\cdot c\cdot 2^k=cl2^{l-1}$,$c$ 是某个常数,因为在每 2^k 步中我们只扫描到区域 k 一次。于是,每趟合并扫描需要 $O(N\log N)$步。由于在平衡合并中有 $O(\log N)$趟扫描,故最坏情况下的完全排序时间肯定是 $O(N(\log N)^2)$;这在渐近意义下比快速排序的最坏情况好得多。

但是,如果我们把这个方法应用到 5.4.6 小节的 100 000 个记录的例子上,它实际上并不是非常好的,因为送往磁带 1 的信息将超出一卷磁带的内容。即使忽略这一事实,而且即使使用关于读/写/计算重叠以及块间间隔长度等最乐观的假定,我们会发现,为完成这一排序,仍将需要大约 37h! 所以这个方法只有纯学术的意义;当 N 是在一个实用的范围时,$O(N(\log N)^2)$中的常数太大了,不能令人满意。

一条磁带排序 我们能否只用一条磁带呢? 不难看出,上面描述的 P 阶冒泡排序能转化成一条磁带的排序,但结果可能是极坏的。

H. B. Demuth[Ph. D. thesis(Stanford University, 1956),85]发现,具有有限内存的一台计算机,当它在磁带上运动有限距离时,它所能减少的一个排列的反序个数,不多于一个有限的数量。因此每个单磁带的排序算法,平均说来至少必须花费 N^2d 个时间单位(其中 d 为某个依赖于计算机的配置正的常数)。

R. M. Karp 以一种非常有趣的方式研究了这个课题,并且发现用一条磁带进行排序的一个最优方式。要讨论 Karp 的算法,最好把这个问题改变如下:*利用单部电梯在各楼层之间运送人们的最快方法是什么*[参见由 Randall Rustin 主编的*Combinatorial Algorithms* (Algorithmics Press,1972),17~21]?

考虑一个 n 层的建筑物,每层上的房间刚好可住 b 个人。这个建筑物没有门、窗、楼梯,但它有可以停于每层上的电梯。在建筑物中有 bn 个人,在每个特定的楼层恰好有 b 个人要去。电梯至多能容 m 个人,而且它从第 i 层到第 $i+1$ 层要花费一个时间单位。如果要求电梯在第一层开始和结束的话,我们希望找出使所有的人都到达适当层的最快方法。

不难看出这个电梯问题和单磁带排序问题之间的联系:人是记录而建筑物是磁带,楼层是磁带上个别的块,而电梯是计算机的内存。一个计算机的程序比一个电梯操作员有更多的灵活性(例如,它可以对人们复制或暂时把他们分成不同层上的

两部分,等等);但下列的解法不必做这样的操作,而以能想像到的最快时间来解决这个问题。

Karp 的算法需要下列两张辅助表:

$$u_k,\ 1 \leqslant k \leqslant n:\ 在 \leqslant k 的层上其目的地 > k 的人数$$
$$d_k,\ 1 \leqslant k \leqslant n:\ 在 \geqslant k 的层上其目的地 < k 的人数 \qquad (4)$$

当电梯空时,对于 $1 \leqslant k < n$,我们总有 $u_k = d_{k+1}$,因为在每层上有 b 个人;在 $\{1,\cdots,k\}$ 层上等待乘电梯的人数,必须等于在 $\{k+1,\cdots,n\}$ 上相应的人数。由定义,$u_n = d_1 = 0$。

显然,对于 $1 \leqslant k < n$,电梯至少必须进行 $\lceil u_k/m \rceil$ 次从层 k 到层 $k+1$ 的运行,因为仅 m 个乘客可以在每次运行中上楼。类似地,它必须至少进行 $\lceil d_k/m \rceil$ 次从第 k 层到 $k-1$ 层的运行,因此对于任何正确的调度,电梯至少需花费

$$\sum_{k=1}^{n} (\lceil u_k/m \rceil + \lceil d_k/m \rceil) \qquad (5)$$

个时间单位。Karp 发现,当 $u_1,\cdots,u_{n-1}$ 非 0 时,这个下限实际上可以达到。

定理 K *如果对于 $1 \leqslant k < n$,$u_k > 0$,则存在一个电梯调度,它在极小时间(5)之下载运每一个人到他的目的地。*

证明　假定在建筑物中有额外的 m 个人;他们开始在电梯中,并且他们的目标层人为地置成 0。电梯可以按照下列算法来进行操作,以 k(当前的层)等于 1 开始:

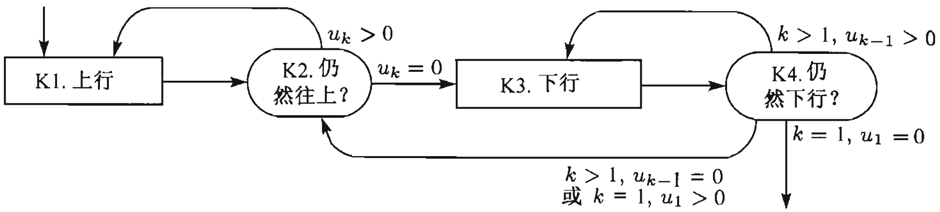


图 87　Karp 的电梯算法

K1. [上行] 请当前在电梯中或在第 k 层上的 $b+m$ 个人中,有最大目的地的那 m 个人上电梯,而其余的人留在 k 层上。

设现在电梯上有 u 个人的目的地 $>k$,另外 d 个人的目的地 $\leqslant k$(结果将是 $u = \min(m, u_k)$;如果 $u_k < m$,则可能要运送某些人离开他们的目的地。这表示他们为整体利益作出牺牲)。u_k 减 u,d_{k+1} 加 d,而后 k 加 1。

K2. [仍然往上?] 如果 $u_k > 0$,则返回步骤 K1。

K3. [下行] 请当前在电梯或在 k 层的 $b+m$ 个人当中,那些具有最小目的地的人进电梯,而其余的停留在 k 层上。

设现在电梯中有 u 个人的目的地 $\geqslant k$,另外 d 个人的目的地 $< k$(结果将总

是 $u=0$ 和 $d=m$，但这里所述的算法仍用一般的 u 和 d 来描述，为的是要使证明更清楚些)。d_k 减 d，u_{k-1}加 u，而后 k 减 1。

K4. [仍然往下?] 如果 $k>1$ 和 $u_{k-1}>0$，则返回步骤 K3。如果 $k=1$ 和 $u_1=0$ 则此算法结束(每个人已经安全到达他的目的地，而 m 个"额外者"也回到电梯中)。否则返回到步骤 K2。

图 88 示出此算法的一个例子，这是一座 9 层建筑物且 $b=2, c=3$。注意，尽管电梯走的是最小的距离，6 人中仍有 1 人被暂时送到第 7 层。在步骤 K4 中测试 u_{k-1}的思想，如同我们将见到的那样，是本算法的关键。

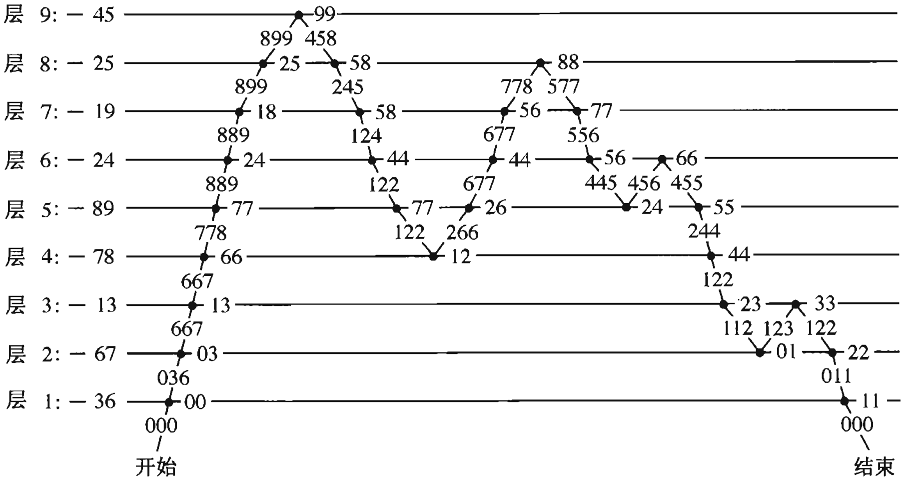


图 88 利用一个小的缓慢的电梯来重新安排人们的最佳方式
(每个人以他的目的层号来表示)

为了证明这个算法的正确性，我们注意，如果我们认为在电梯里的人是在"当前"层 k 上，则步骤 K1 和 K3 总是保持 u 和 d 表(4)的内容是最新的。现在有可能用归纳法证明下列性质在每个步骤的开始时均成立:

$$u_l=d_{l+1} \qquad \text{对于 } k\leqslant 1<n \tag{6}$$

$$u_l=d_{l+1}-m \qquad \text{对于 } 1\leqslant l<k \tag{7}$$

$$u_{l+1}=0 \qquad \text{如果 } u_l=0 \text{ 和 } k\leqslant l<n \tag{8}$$

进而，在步骤 K1 的开始，在$\leqslant k$ 层上且目的地$>k$ 的所有人当中，具有最高目的地的 $\min(u_k,m)$个人，是在电梯中或在 k 层上。在步骤 K3 的开始，在$\geqslant k$ 层且目的地$<k$ 的所有人当中，具有最低目的地的 $\min(d_k,m)$个人，是在电梯中或在 k 层上。

从这些性质得出，步骤 K1 和 K3 中括号内的注释是正确的。因此，步骤 K1 的每一次执行都使$\lceil u_k/m\rceil$减少 1，而保持$\lceil d_{k+1}/m\rceil$不变；步骤 K3 的每次执行都使

$\lceil d_k/m\rceil$减少 1，而保持$\lceil u_{k-1}/m\rceil$不变。因此这个算法必定在有限步之内结束，而且每个人由于(6)和(8)，必然处于正确的楼层上。 ▌

当 $u_k=0$ 和 $u_{k+1}>0$ 时，我们有一个"断开"的状态；尽管没有人要从$\leqslant k$ 的层上升到$\geqslant k+1$ 的层，电梯还是必须升到 $k+1$ 层以便重新安排那儿的人。不失一般性，我们可以假定 $u_{n-1}>0$；于是每个正确的电梯调度都必须至少包括

$$2\sum_{1\leqslant k<n}\max(1,\lceil u_k/m\rceil) \tag{9}$$

次移动，因为我们要求电梯返回到层 1。达到这一下限的调度是容易构造出来的(习题 4)。

习　题

1. [*20*] 正文中讨论的 P 阶冒泡排序，仅仅使用向前读和重绕。能否把这个算法修改成利用向后读？

2. [*M26*] 求出在(3)中定义的数 X_N，Y_N 的明显的"封闭形式"的解[提示：分析等式 5.2.2-(19)的解]。

3. [*38*] 是否有仅仅基于键码(不是数字性质的)比较的两条磁带的排序方法，当对 N 个记录进行排序时，它的磁带运动在最坏的情况下为 $O(N\log N)$[快速排序平均可达到这一点，但在最坏情况下不行，而 Hennie-Stearns 方法(图 86)则达到 $O(N\log N)^2)$]？

4. [*M23*] 在电梯问题中，假设有下标 p 和 q，且 $q\geqslant p+2$，$u_p>0$，$u_q>0$，以及 $u_{p+1}=\cdots=u_{q-1}=0$。说明怎样构造至多需要(9)时间单位的一个调度。

▶5. [*M23*] 真或假：定理 K 中，在算法的步骤 K1 之后，电梯中人的目的地均高于楼层$<k$ 的所有人的目的地。

6. [*M30*] (R. M. Karp)　把电梯问题(图 88)推广到下列情形：对于 $1\leqslant j\leqslant n$，开始时在 j 层上有 b_j' 个乘客，以及有 b_j' 个乘客其目的地是 j 层。证明存在一个调度，它花费 $2\sum_{k=1}^{n-1}\max(1,\lceil u_k/m\rceil,\lceil d_{k+1}/m\rceil)$ 个时间单位，且在任何时刻绝不允许在层 j 上有多于 $\max(b_j,b_j')$个乘客[提示：如果必要，引进虚构的人，使得对于所有的 j，$b_j=b_j'$]。

7. [*M40*] (R. M. Karp)　推广习题 6 的问题，用一个客车通过的磁道路网络来代替电梯所遵循的线性通路，其中假定该网络形成任意的自由树。这个客车容量有限，而且所希望的是，客车只行驶最小的距离，就把旅客运送到他们的目的地。

8. [*M32*] 设正文所讨论的电梯问题中，$b=1$。问 n 层上的 n 个人有多少种排列，使得在(4)中，对于 $1\leqslant k\leqslant n$，有 $u_k\leqslant 1$[例如，3 1 4 5 9 2 6 8 7 就是一种这样的排列]？

▶9. [*M25*] 找出 5.2.2 小节中图 16 描述的"鸡尾混合排序"和在 $b=1$ 时(4)的数 $u_1,u_2,\cdots,u_n$ 之间的重要联系。

10. [*20*] 如何仅用两条磁带来对多卷文件排序？

*5.4.9　磁盘和磁鼓

前面我们一直把磁带作为外部排序的工具，但是通常还有更为灵活的大容量存

储设备可用。尽管这样的“海量存储”或者“直接存取存储”设备形式五花八门，但它们可以粗略地以如下一些性质来表征：

i) 被存储信息的任何特定部分都可快速地访问。

ii) 在内存和外存之间可以迅速地传输连续的字块。

磁带满足 ii)但不满足 i)，因为它要花费较长时间来从磁带的一端达到另一端。

在对一种外部存储编写大量程序之前，都应当仔细地研究它具有的一些特性。但是技术的变化如此迅速，以致在这里不可能对所有可利用的形形色色的硬件进行完备的讨论。因此，我们将仅仅考虑某些典型的存储设备，这些设备展示了对于排序问题有用的方法。

满足 i)和 ii)的最普遍的外部存储类型之一，为一个磁盘设备(见图 89)。数据被存放在一些急速转动的圆磁盘上，这些圆磁盘上涂了磁性材料；一个像梳子样的存取臂对于每个磁盘面包含一个或多个“读/写头”，它用来存储和检索信息。每个磁盘面分成称做为*磁道*的同心的环，使得每当这个磁盘完成一次转动时，整个磁道的数据就通过一次读/写头。存取臂可以伸进和伸出，从一个磁道到另一个磁道地移动读/写头；但这种运动花费时间。可以不必重新定位存取臂就能读或写的一组磁道，称为一个*柱面*。例如，图 89 示出了一个磁盘设备，它的每个平面恰有一个读/写头；虚圆圈表示一个柱面，它由当前正被读/写头扫描的所有磁道所组成。

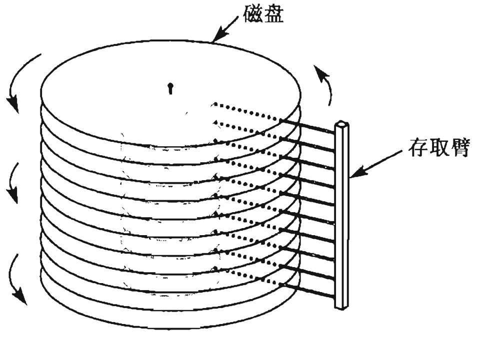


图 89 一个磁盘机

为了叙述得更确切一点儿，我们考虑假想的 MIXTEC 磁盘组设备，对于它

1 个磁道 =5000 个字符

1 个柱面 =20 个磁道

1 个磁盘设备 =200 个柱面

这样一个磁盘设备含有 2 千万个字符，略少于一条 MIXT 磁带上所能存储的数据量。在某些机器上，接近中心的磁道的字符比接近边缘的磁道要少；这势必使程序设计更为复杂，幸而，MIXTEC 避免了这样的问题(参见 5.4.6 小节关于 MIXT 磁带的讨论。如同在该小节那样，我们通过考虑对 20 世纪 70 年代早期来说典型的那些机器特征，来研究典型的技术；现代的磁盘要大得多，也快得多)。

在一个磁盘设备上为读和写所需要的时间实质上是 3 个量之和：

·寻道时间(为把存取臂移动到适当的柱面所需时间)。

·等待时间(读/写头到达正确位置之前的转动延迟)。

·传输时间(数据通过读/写头时的转动延迟)。

在 MIXTEC 设备上为从柱面 i 转移到柱面 j 所需要的寻道时间为 $25+\frac{1}{2}|i-j|$

ms。如果 i 和 j 是在 1 和 200 之间随机选择的整数，则 $|i-j|$ 的平均值为 $2\binom{201}{3}/200^2\approx66.7$，所以平均寻道时间大约是 60ms。MIXTEC 磁盘每 25ms 转动一圈，所以等待时间平均约 12.5ms。n 个字符的传输时间是 $(n/5000)\times25\text{ms}=5n\text{ms}$ $\left(\text{这大约是 5.4.6 小节的例子所用 MIXT 磁带的传输速度的 } 3\frac{1}{3}\text{倍}\right)$。

因此，对于排序来说，MIXTEC 磁盘和 MIXT 磁带之间的主要差别是：

a)磁带仅可被顺序地存取；

b)个别的磁盘操作势必要求相当多的开销（寻道时间 + 相对于停止/启动时间的等待时间）；

c)磁盘传输速度比较快。

通过使用更灵巧的磁带的合并型式，能够对缺点 a)做些补偿。我们现在的目标是考虑能补偿缺点 b)的某些灵巧的磁盘排序算法。

克服等待时间 让我们首先考虑使延迟极小化的问题。延迟是由这样一个事实引起的，当启动一条输入输出指令时，磁盘往往并不在适当的位置上。我们不能使磁盘旋转得更快，但可以应用某些技巧，来减少甚至消除所有的等待时间。加上更多的存取臂显然是有帮助的，但这将是一个昂贵的硬件修改。下面是某些“软件”的思想：

1)如果一次读或写同一个柱面的若干磁道，则我们就避免了除了第一次以外所有磁道上的等待时间（以及寻道时间）。一般地说，通常都有可能以这样一种方式来同步计算时间和磁盘运动的时间，即可以无须等待延迟而执行一系列的输入输出指令。

2)考虑读半个数据磁道的问题（见图 90）：如果当磁头在点 A 时开始读命令，则没有等待延迟，因而用于读的总时间恰是传输时间 $\frac{1}{2}\times25\text{ms}$。如果当磁头在点 B 时开始这命令，则我们需要转 $\frac{1}{4}$ 圈以用于等待和转 $\frac{1}{2}$ 圈用于传输，总共 $\frac{3}{4}\times25\text{ms}$。当第一开始位于 C 时出现最有趣的情况：通过适当的硬件和软件，我们不需要浪费 $\frac{3}{4}$ 的转动用于等待延迟。立即可开始读，读到输入缓冲区的第二半；然后在 $\frac{1}{2}\times25\text{ms}$ 的中止之后，可以继续读到这个缓冲区的头一半，于是当再次到达轴 C 时，指令已告完成。以一种类似的方式，我们可以确保总共的等待 + 传送时间决不超过转一圈的时间，而不管磁盘的初始位置如何。通过这个方案减少了平均等待延迟时间，如果我们正在读或写一个磁道的指定部分 x 时

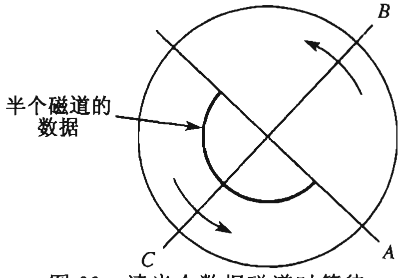


图 90 读半个数据磁道时等待时间的分析

$(0<x\leqslant 1)$,可由延迟半圈减少到延迟$\frac{1}{2}(1-x^2)$圈。当正在读或写整个一个磁道$(x=1)$时,这个技术消除了所有等待时间。

磁鼓:无寻道的情况　某些外部存储设备,传统上称为磁鼓存储,由于对每个磁道都有一个读/写头,从而消除了寻道时间。如果图 90 的技术被应用到这样的设备上,则寻道时间和等待时间都将减少为 0,其中假定我们总是一次读或写一个磁道;这时传输时间是惟一的限制因素,这是一种理想情况。

让我们再次考虑 5.4.6 小节的应用实例,用有 100 000 个字符的内存对 100 000 个每个有 100 个字符的记录排序。有待排序的数据总量填满了一个 MIXTEC 盘的一半。通常不可能同时在单个磁盘设备上进行读和写;我们将假定有两个磁盘可以利用,使得读和写可以彼此重叠。事实上,暂时我们将假定“磁盘”实际上是磁鼓,它包含 4000 个每个含 5000 个字符的磁道,而且不需要寻道时间。

应使用什么排序算法呢?合并的方法是一个相当自然的选择,内部排序的其它方法不见得更利于在磁盘上实现,5.2.5 小节的基数技术除外。但是 5.4.7 小节的考虑表明,对于一般应用,基数排序通常比合并要低劣,因为该小节的对偶性定理既可应用于磁带,也可应用于磁盘。然而,基数排序在键码为一致分布的,以及在可以并行地使用磁盘时确实有很大的优点,因为通过键码的最高有效位的初始分布,将把工作分成为不需要进行进一步通信的一些独立的子程序(例如,参见 R.C. Agarwal,*SIGMOD. Record* **25** ,2(June1996),240～246)。

在以下的讨论中,我们将专注于合并排序。

为了开始对上述问题的一个合并排序,我们可以使用替代选择,用两个 5000 个字符的输入缓冲区和两个 5000 字符的输出缓冲区。事实上,如果用从选择树来的记录代替当前输入缓冲区中的记录,则有可能把上述缓冲区减少为 3 个 5000 字符的缓冲区。这把 85000 个字符(850 个记录)送到一株选择树,所以对于我们所例举数据的一次扫描将形成大约 60 个初始路段(见等式 5.4.6-(3))。如果假定内部处理时间足够快,能赶上输入输出的速度,即每 500μs 一个记录移动到输出缓冲区,则这次扫描仅仅花费大约 50s 的时间。如果有待排序的输入出现在一条 MIXT 磁带上,而不是出现在一个磁鼓上,则这个扫描将慢一些,这是因磁带的速度所致。

对于两个磁鼓和全磁道的读/写,不难看到,如果我们令 P 尽可能地大,则 P 路合并的总传输时间即为极小。可惜,我们不能对所有初始路段简单地做一个 60 路的合并,因为在内存中没有 60 个缓冲区空间(少于 5000 个字符的一个缓冲区将导致不必要的等待时间)。如果进行 P 路合并,把所有数据从一个磁鼓传输到另一个磁鼓,使得读和写重叠,则合并扫描的次数为$\lceil\log_P 60\rceil$,所以如果 $8\leqslant P\leqslant 59$,我们就可以在两次扫描中来完成这项工作。最小的这样的 P 减少了内部计算的量,所以我们选择 $P=8$;如果已经形成 65 个初始路段,则将取 $P=9$。如果已经形成了 82 个或更多的初始路段,则将取 $P=10$,但由于仅仅有 18 个输入缓冲区和 2 个输出缓冲区的空间,所以在合并期间将有挂起的可能性(见算法 5.4.6F);在这样一种情况下,可能更宜于对一小部分

数据做部分的扫描，并且把初始路段的数目减到 81 或更少。

在我们的假定之下，两个合并扫描都将花费大约 50s，所以在这种理想的情况下完成整个的排序仅用 2.5min（加上用于簿记操作、初始化等等的几秒钟）。这比 5.4.6 小节中所考虑的最好的 6 磁带排序快 6 倍；速度提高的原因是：改进了外部/内部传输速度（快 3.5 倍），更高阶的合并（我们不能做 8 路磁带合并，除非有 9 条或更多的磁带），以及输出仍保持在磁盘上（最后的重绕等不必要）。如果初始的输入和排序的输出要求在 MIXT 磁带上，而磁鼓仅用作合并，则相应的排序时间将大约为 8.2min。

如果仅仅有一个磁鼓可利用而不是两个，则输入输出时间将花费两倍之长，因为读和写必须分开来完成（事实上，输入输出操作将花费 3 倍时间，因为我们将冲掉初始的输入数据；在这样一种情况下，如果硬件不提供对已写信息的自动校验，则宜于在每一个写操作之后紧接着一个“向后读校验”操作，免得某些输入数据丢失以致无法检索）。但由于我们可以使用部分扫描的方法，此方法处理一部分数据记录的次数多于另一部分，因此就可以挽回一些时间上的损失。两个磁鼓的情况需要偶数次或奇数次地处理所有数据，但一个磁鼓的情况可以使用更一般的合并型式。

我们在 5.4.4 小节就注意到，合并型式可以通过树来表示，而且对应于一个合并型式的传输时间同其树的外部路径长度成比例。只有某些树（T-lifo 或者强 T-fifo）可以用作有效的磁带合并型式，因为某些路段在进行合并过程中会掩蔽在一条磁带的中间。但是在磁盘或磁鼓上，所有树都定义了可用的合并型式，只要它们内部节点的层次对于可利用的内部存储的大小来说，不是太大即可。

因此，我们可以通过选择一株具有极小外部路径长度的树来使传输时间极小化，诸如一株完备的 P 叉树，这里 P 尽可能地大。由等式 5.4.4-(9)，如果这样一株树有 S 个外部节点（叶），则它的外部路径长度等于

$$qS - \lfloor (P^q - S)/(P-1) \rfloor \qquad q = \lceil \log_P S \rceil \tag{1}$$

设计一个按照完备的 P 叉树型式来合并的算法，是特别容易的。例如，图 91 所示为 $P=3, S=6$ 的情况。首先，如果必要的话，我们加上“虚拟路段”，使得 $S \equiv 1$ (modulo$P-1$)；然后，按照“-fifo”的规则来组合路段，在每个阶段，把队列前 P 个“最老的”的路段合并成为一个路段，并把该路段放在后边。

如果所有的初始路段都有相同的长度，则完备的 P 叉树给出一个最优的型式，但是如果某些路段比其余的长，则我们常常可以做得更好。这种一般情况的最优型

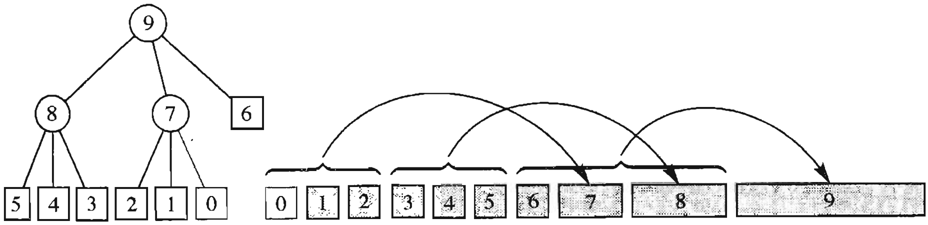


图 91　具有 6 片叶的完备三叉树及其相应的合并型式

式,可以毫无困难地使用 Huffman 方法来加以构造(习题 2.3.4.5-10),它可以用合并的语言叙述如下:"首先加上 $(1-S)\bmod(P-1)$ 个长度为 0 的虚拟路段,然后反复地把 P 个最短的现存路段合并在一起,直到剩下一个路段为止。"当所有路段都有相同长度时,这个方法就归结为前面所述的-fifo 规则。

在 100 000 个记录的例子中,我们可以作九路合并,因为内存中可以放下 18 个输入缓冲区和两个输出缓冲区,而且算法 5.4.6F 将重叠所有的计算时间。如果所有初始路段都有相同的长度,则完备的具有 60 片叶子的 9 叉树对应于具有 $1\frac{29}{30}$ 趟扫描的一个合并型式。因此,若在每次写之后使用向后读校验,则使用一个磁鼓的总排序时间,约等于 7.4min。更高的 P 值可以稍稍减少这个运行时间;但当缓冲区变得太满或太空时,由于可能出现"读挂起",因此情况就变得复杂了。

寻道时间的影响 以上的讨论表明,对于磁鼓来构造"最优"合并型式是比较容易的,因为寻道时间和等待时间实际上都不存在。但当使用有小的缓冲的磁盘时,通常花费在寻找信息的时间比起读信息的时间要长,所以寻道时间对于排序策略有相当大的影响。减少合并的阶数 P,就可能使用更大的缓冲区,所以就需要较少的寻道时间;这通常能弥补较小的 P 值所引起的额外的传输时间。

寻道时间同存取臂通过的距离有关,因此我们可以尝试把事情安排成使得这个距离成为极小。例如,首先在柱面内对记录排序可能是明智的。但是大型的合并往往要求在柱面之间作大量的跳跃(参见习题 2)。其次,现代操作系统的多程序设计的能力意味着,用户正失去对磁盘存取臂的控制;因此,我们通常认为,假定每一个磁盘命令都包含一个"随机"的寻道是合理的。

我们的目标是发现一个合并型式,它在寻道时间和传输时间之间实现最好的平衡;为此目的,需要某种方法,相对于一个特定的硬件配置来估计任何特定树的好坏。例如,考虑图 92 中的树;我们要评估为进行相应的合并它将花费多长时间,以便能把这株树同其它树作比较。

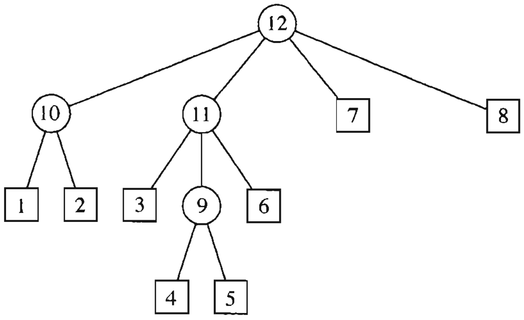


图 92 外部路径长度为 16 而叉数路径长度为 52 的一株树

在下面的讨论中,我们将对磁盘合并作某些简单的假定,以便说明某些一般的思想。假定:(i)为了读或者写 n 个字符要花费 $72.5+0.005n$ms;(ii)内存中有 100 000 个字符的位置可用作工作存储;(iii)为把每个字符从输入传输到输出,需要平均 0.004ms 的计算时间;(iv)在读、写或计算之间没有重叠;(v)用作输出的缓冲区大小不必和下次扫描时用作读数据的缓冲区大小一样。在这些简单假定之下对排序问题的分析,将为我们转向更复杂的情况提供某些启示。

如果进行一个 P 路合并，则我们可以把内部工作存储分成为 $P+1$ 个缓冲区域，P 个作为输入和 1 个作为输出，而每个缓冲区都有 $B=100\ 000/(P+1)$ 个字符。假定正在合并的文件总共包含 L 个字符；于是我们将做近似于 L/B 个输出操作和大约相同数量的输入操作，所以在我们的假定之下总共的合并时间将近似于

$$2\left(72.5\frac{L}{B}+0.005L\right)+0.004L=(0.00145P+0.01545)L\ \text{ms} \tag{2}$$

换言之，L 个字符的 P 路合并将花费大约 $(\alpha P+\beta)L$ 个单位时间，其中 α,β 为依赖于寻道时间、等待时间、计算时间以及内存大小的常数。这个公式导致了一个有趣的、构造一个好的磁盘合并型式的方法。例如，考虑图 92，并假定所有的初始路段（以正方形“叶”节点表示）长度都为 L_0，则在节点 9 和 10 处的合并各需花费 $(2\alpha+\beta)(2L_0)$ 个单位时间，在节点 11 处的合并花费 $(3\alpha+\beta)(4L_0)$ 个，在节点 12 的最后合并花费 $(4\alpha+\beta)(8L_0)$ 个。因此，总共的合并时间为 $(52\alpha+16\beta)L_0$ 单位。这里的系数“16”是我们所熟知的，它只不过是该树的外部路径长度。然而，α 的系数“52”是一个新的概念，它可以称为树的*叉数路径长度*（degree path length）；它是对于所有叶节点取的从叶到根的通路上内部节点叉数的和。例如，在图 92 中，叉数路径长度为

$$(2+4)+(2+4)+(3+4)+(2+3+4)+(2+3+4)$$
$$+(3+4)+(4)+(4)=52$$

如果 $\mathcal{T}$ 是任意树，则令 $D(\mathcal{T})$，$E(\mathcal{T})$ 分别表示它的叉数路径长度和外部路径长度。我们的分析可以综述如下：

定理 H　*如果对于 L 个字符进行一个 P 路合并所需要的时间是 $(\alpha P+\beta)L$，且如果有 S 个等长路段有待合并，则最好的合并型式对应于在有 S 个叶的所有树中，使 $\alpha D(T)+\beta E(\mathcal{T})$ 为极小的一株树 $\mathcal{T}$。*

此定理隐含于一篇未发表的论文中，George U. Hubbard 在 1963 年的 *ACM* National Conference 上介绍过它。

令 α 和 β 是固定的常数；如果一株树对于具有相同叶数的所有树 $\mathcal{T}$ 有极小的 $\alpha D(\mathcal{T})+\beta E(\mathcal{T})$ 值，则我们说这株树是*最优的*。不难看出，一株最优树的所有子树是最优的。因此我们可以通过把具有 $<n$ 片叶的最优树合在一起，构造出具有 n 片叶的最优树。

定理 K　*对于 $1\leqslant m\leqslant n$，通过规则*

$$A_1(1)=0 \tag{3}$$

$$A_m(n)=\min_{1\leqslant k\leqslant n/m}(A_1(k)+A_{m-1}(n-k))\quad 对于\ 2\leqslant m\leqslant n \tag{4}$$

$$A_1(n)=\min_{2\leqslant m\leqslant n}(\alpha mn+\beta n+A_m(n))\quad 对于\ n\geqslant 2 \tag{5}$$

定义数 $A_m(n)$ 的序列。现在所有具有 n 片叶的树 $\mathcal{T}$ 中，$A_1(n)$ 是 $\alpha D(\mathcal{T})+\beta E(\mathcal{T})$ 的极小值。

证明 等式(4)蕴涵着,$A_m(n)$是对于所有使得 $n_1+\cdots+n_m=n$ 的正整数 n_1,$\cdots$,n_m,和式 $A_1(n_1)+\cdots+A_1(n_m)$的极小值。对 n 用归纳法即得结果。 ▌

递归关系(3),(4),(5)也可用来构造最优树本身:令 $k_m(n)$是在 $A_m(n)$的定义中使极小值出现的一个值,则我们可以通过在根处加入 $m=k_1(n)$个子树,构造具有 n 片叶子的一株最优树;这些子树又是分别具有 $k_m(n)$,$k_{m-1}(n-k_m(n))$,$k_{m-2}(n-k_m(n)-k_{m-1}(n-k_m(n)))$,…片叶子的最优树。

例如,表 1 说明了当 $\alpha=\beta=1$ 时的这个构造。在该表的右边是对应的最优树的简单说明。例如,当 $n=22$ 时,条目“4:9:9”意味着,具有 22 片叶子的最优树 $\mathcal{T}_{22}$,可以通过把 $\mathcal{T}_4$,$\mathcal{T}_9$ 和 $\mathcal{T}_9$ 组合在一起来得到(见图 93)。最优树不是惟一的;例如,5:8:9 同 4:9:9 一样地好。

对(2)的推导表明,当使用 $P+1$ 个相等的缓冲区域时,关系式 $\alpha\leqslant\beta$ 总成立。当要求寻道时间极小化,而不考虑传输时间时,即出现表 1 和图 93 中的极限情况 $\alpha=\beta$。

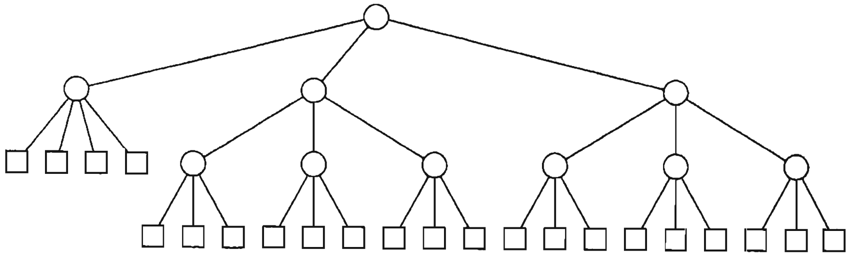


图 93 当在定理 H 中 $\alpha=\beta$ 时,合并 22 个相等长度的初始路段的一种最优方式。在与正文中的等式(2)同样的假定之下,这个型式使寻道时间极小化

表 1 当 $\alpha=\beta=1$ 时最优树特征 $A_m(n)$,$k_m(n)$

	m												
n	1	2	3	4	5	6	7	8	9	10	11	12	树	n
1	0,0												—	1
2	6,2	0,1											1:1	2
3	12,3	6,1	0,1										1:1:1	3
4	20,4	12,1	6,1	0,1									1:1:1:1	4
5	30,5	18,2	12,1	6,1	0,1								1:1:1:1:1	5
6	42,2	24,3	18,1	12,1	6,1	0,1							3:3	6
7	52,3	32,3	24,1	18,1	12,1	6,1	0,1						1:3:3	7
8	62,3	40,4	30,2	24,1	18,1	12,1	6,1	0,1					2:3:3	8
9	72,3	50,4	36,3	30,1	24,1	18,1	12,1	6,1	0,1				3:3:3	9
10	84,3	60,5	44,3	36,1	30,1	24,1	18,1	12,1	6,1	0,1			3:3:4	10
11	96,3	72,4	52,3	42,2	36,1	30,1	24,1	18,1	12,1	6,1	0,1		3:4:4	11
12	108,3	82,4	60,4	48,3	42,1	36,1	30,1	24,1	18,1	12,1	6,1	0,1	4:4:4	12

（续）

	叉													
n	1	2	3	4	5	6	7	8	9	10	11	12	树	n
13	121,4	92,4	70,4	56,3	48,1	42,1	36,1	30,1	24,1	18,1	12,1	6,1	3:3:3:4	13
14	134,4	102,5	80,4	64,3	54,2	48,1	42,1	36,1	30,1	24,1	18,1	12,1	3:3:4:4	14
15	147,4	114,5	90,4	72,3	60,3	54,1	48,1	42,1	36,1	30,1	24,1	18,1	3:4:4:4	15
16	160,4	124,7	102,4	80,4	68,3	60,1	54,1	48,1	42,1	36,1	30,1	24,1	4:4:4:4	16
17	175,4	134,8	112,4	90,4	76,3	66,2	60,1	54,1	48,1	42,1	36,1	30,1	4:4:4:5	17
18	190,4	144.9	122,4	100,4	84,3	72,3	66,1	60,1	54,1	48,1	42,1	36,1	4:4:5:5	18
19	205,4	156.9	132,5	110,5	92,3	80,3	72,1	66,1	60,1	54,1	48,1	42,1	4:5:5:5	19
20	220,4	168,9	144,4	120,5	100,4	88,3	78,2	72,1	66,1	60,1	54,1	48,1	5:5:5:5	20
21	236,5	180,9	154,4	132,4	110,4	96,3	84,3	78,1	72,1	66,1	60,1	54,1	4:4:4:4:5	21
22	252,3	192,10	164,4	142,4	120,4	104,3	92,3	84,1	78,1	72,1	66,1	60,1	4:9:9	22
23	266,3	204,11	174,5	152,4	130,4	112,3	100,3	90,2	84,1	78,1	72,1	66,1	5:9:9	23
24	282,3	216,12	186,5	162,5	140,4	120,4	108,3	96,3	90,1	84,1	78,1	72,1	5:9:10	24
25	296,3	229,12	196,7	174,4	150,5	130,4	116,3	104,3	96,1	90,1-	84,1	78,1	7:9:9	25

回到最初的应用，我们尚未考虑如何首先得到初始的路段；如果没有读/写/计算的重叠，则替代选择就失去了它的某些优点。也许我们应该填满整个内存，然后排序并输出结果；每一个这样的输入和输出操作都只需要一次寻道。或者也许拿出比如说内存的 20％作为一个综合的输入/输出缓冲区，并进行替代选择更好。这要求 5 倍的寻道时间（一个额外的 60s 左右时间！），但它使初始路段的数目从 100 减少到 64；如果输入文件原来就是相当有序的，则减少就更为显著。

如果我们决定不使用替代选择，则对于 $S=100$，$\alpha=0.00145$，$\beta=0.01545$ 的最优树［参见(2)］是相当平凡的：它只不过是对数据在两次扫描中完成的一个 10 路合并。如果允许对内部排序花费 30s（比如说，100 次快速排序），则初始分配扫描大约花费 2.5min，每次合并扫描将近花费 5min，总共是 12.4min。如果我们决定使用替代选择，则对于 $S=64$ 的最优树也同样是平凡的（两个 8 路合并扫描）；初始分布扫描大约花费 3.5min，每次合并扫描大约花费 4.5min，估计总共的时间为 12.6min。别忘了，这两个方法实际上都放弃了所有的读/写/计算的重叠，为的是有更大的缓冲区，并减少寻道时间。这些估计的时间均未包括为向后读检验操作所可能需要的时间。

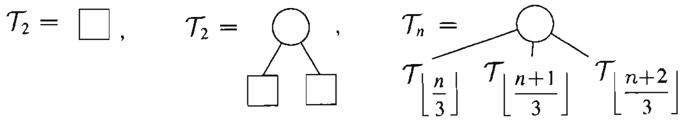
实际上，最后的合并扫描与其它几次扫描往往十分不同；例如，输出常常被展开和/或写到磁带上。在这种情况下，树型式应该利用在根处的另一种最优性准则来加以选择。

***对最优树的进一步考察**　尽管实际情况参数通常满足 $0\leqslant\alpha\leqslant\beta$，但是考察定理 H 和 K 中的极端情况 $\beta=0$ 是有趣的。具有 n 片叶子的树什么样的有最小的叉数路径长度？奇怪的是，结果证明三路合并是最好的。

定理 L　*一株具有 n 片叶子的树，其叉数路径长度决不小于*

$$f(n)=\begin{cases}3qn+2(n-3^q) & \text{如果 } 2\cdot 3^{q-1}\leqslant n\leqslant 3^q\\ 3qn+4(n-3^q) & \text{如果 } 3^q\leqslant n\leqslant 2\cdot 3^q\end{cases}\tag{6}$$

由下列规则



(7)

定义的三叉树 $\mathcal{T}_n$ 有极小叉数路径长度。

证明 重要的是要注意 $f(n)$ 是一个凸函数,即

$$f(n+1)-f(n)\geqslant f(n)-f(n-1)\qquad \text{对于所有 } n\geqslant 2\tag{8}$$

这个性质的正确性由下列引理得出,而这个引理是习题 2.3.4.5-17 结果的对偶。

引理 C. *在正整数上定义的一个函数 $g(n)$ 满足*

$$\min_{1\leqslant k<n}(g(k)+g(n-k))=g(\lfloor n/2\rfloor)+g(\lceil n/2\rceil),\quad n\geqslant 2\tag{9}$$

当且仅当它是凸的。

证明 如果对于某个 $n\geqslant 2$, $g(n+1)-g(n)<g(n)-g(n-1)$,我们有 $g(n+1)+g(n-1)<g(n)+g(n)$,同(9)相冲突。反之,如果对于 g,(8)成立,而且如果 $1\leqslant k<n-k$,由凸性,我们有 $g(k+1)+g(n-k-1)\leqslant g(k)+g(n-k)$。 ▌

引理 C 证明的后面部分可推广到任何 $m\geqslant 2$ 以证明每当 g 为凸时.

$$\min_{\substack{n_1+\cdots+n_m=n\\ n_1,\cdots,n_m\geqslant 1}}(g(n_1)+\cdots+g(n_m))=$$
$$g(\lfloor n/m\rfloor)+g(\lfloor(n+1)/m\rfloor)+\cdots+g(\lfloor(n+m-1)/m\rfloor)\tag{10}$$

令

$$f_m(n)=f(\lfloor n/m\rfloor)+f(\lfloor(n+1)/m\rfloor)+\cdots+f(\lfloor n+m-1)/m\rfloor)\tag{11}$$

通过证明 $f_3(n)+3n=f(n)$ 以及对于所有 $m\geqslant 2$, $f_m(n)+mn\geqslant f(n)$,便完成了定理 L 的证明(见习题 11)。 ▌

如果最优树总可以像在定理 L 中那样利落地表征,那将是非常好的。但在表 1 中对于 α=β,我们已经看到的结果表明,函数 $A_1(n)$ 并不总是凸的。事实上,表 1 足以否定关于最优树的大多数简单的猜测!但是,在一般情况下我们可以部分地拯救定理 L;M. Schlumberger 和 J. Vuillemin 已经证明,可以避免合并大的阶。

定理 M. *如定理 H 中那样给定 α 和 β,存在一株最优树,其中每个节点的叉数至多为*

$$d(\alpha,\beta)=\left\lceil\min_{k\geqslant 1}\left(k+\left(1+\frac{1}{k}\right)\left(1+\frac{\beta}{\alpha}\right)\right)\right\rceil\tag{12}$$

证明 令 $n_1,\cdots,n_m$ 是使得 $n_1+\cdots+n_m=n$, $A(n_1)+\cdots+A(n_m)=A_m(n)$

的正整数，且 $n_1\leqslant\cdots\leqslant n_m$。另外假定 $m\geqslant d(\alpha,\beta)+1$。令 k 是极小化(12)的值；我们将证明

$$\alpha n(m-k)+\beta n+A_{m-k}(n)\leqslant \alpha nm+\beta n+A_m(n) \tag{13}$$

因此(4)中的极小值对于某个 $m\leqslant d(\alpha,\beta)$总是可达到的。

由定义，由于 $m\geqslant k+2$，我们必定有

$$\begin{aligned}A_{m-k}(n)\leqslant A_1(n_1+\cdots+n_{k+1})+A_1(n_{k+2})+\cdots+A_1(n_m)\leqslant\\ \alpha(n_1+\cdots+n_{k+1})(k+1)+\beta(n_1+\cdots+n_{k+1})+A_1(n_1)+\cdots+A_1(n_m)=\\ (\alpha(k+1)+\beta)(n_1+\cdots+n_{k+1})+A_m(n)\leqslant\\ (\alpha(k+1)+\beta)(k+1)n/m+A_m(n)\end{aligned}$$

因此现在容易得出(13)(仔细观察这个证明可发现，由于某些最优树一定有叉数为 $d(\alpha,\beta)$的节点，所以(12)可能是最好的；参见习题 13)。 ▌

对于 $1\leqslant m\leqslant n\leqslant N$，定理 K 的构造需要 $O(N^2)$个存储单元和 $O(N^2\log N)$个步骤来计算 $A_m(n)$。定理 M 表明仅仅需要 $O(N)$个单元和 $O(N^2)$个步骤。Schlumberger 和 Vuillemin 发现了最优树的一些非常有趣的性质[*Acta Informatica* **3** (1973)，25～36]。其次，如习题 9 所示，可以计算出 $A_1(n)$的渐近值。

***分配缓冲区的另一种方式**　David E. Ferguson[*CACM* **14**(1971)，476～478]指出，如果我们不把所有缓冲区都做成相同大小，则寻道时间即可减少。同样的思想大约同一时间也出现在其他人的文章中[S. J. Waters，*Comp. J.* **14**(1971)，109～112；Ewing S. Walker，Software Age **4**(Auguest-September，1970)，16～17]。

假设我们正在对具有相等长度 L_0 的路段进行四路合并，且内存可容纳 M 个字符。如果把内存划分成相同的缓冲区大小 $B=M/5$，则对于每个输入文件需要大约 L_0/B 次寻道，对于输出需要 $4L_0/B$ 次寻道，加起来为 $8L_0/B=40L_0/M$ 次寻道。但如果我们用 4 个大小为 $M/6$ 的输入缓冲区和 1 个大小为 $M/3$ 的输出缓冲区，则只需要大约 $4\times(6L_0/M)+4\times(3L_0/M)=36L_0/M$ 次寻道！在两种情况下传输时间是相同的，所以由这个改动，我们并不受任何损失。

一般地说，假设要把长度为 $L_1,\cdots,L_p$ 的排序文件合并成为长度为 $L_{p+1}=L_1+\cdots+L_P$ 的一个排序文件，而且假定一个大小为 B_k 的缓冲区被用于第 k 个文件。于是有

$$B_1+\cdots+B_P+B_{P+1}=M \tag{14}$$

其中 M 是可利用的内存的总容量，寻道的次数将近似为

$$\frac{L_1}{B_1}+\cdots+\frac{L_P}{B_P}+\frac{L_{P+1}}{B_{P+1}} \tag{15}$$

现在让我们在条件(14)的约束下尝试使这个量极小化，为方便起见假定 B_k 不必为整数。如果对 B_j 增加 δ 而对 B_k 减少相同的小量，则寻道次数的改变为

$$\frac{L_j}{B_j+\delta}-\frac{L_j}{B_j}+\frac{L_k}{B_k-\delta}-\frac{L_k}{B_k}=\left(\frac{L_k}{B_k(B_k-\delta)}-\frac{L_j}{B_j(B_j+\delta)}\right)\delta$$

所以如果 $L_j/B_j^2 \neq L_k/B_k^2$，这个分配就能改进。因此，仅当

$$\frac{L_1}{B_1^2} = \cdots = \frac{L_P}{B_P^2} = \frac{L_{P+1}}{B_{P+1}^2} \tag{16}$$

时我们得到极小的寻道次数。由于存在一个极小，因此当

$$B_k = \sqrt{L_k}M/(\sqrt{L_1} + \cdots + \sqrt{L_{P+1}}), \quad 1 \leqslant k \leqslant P+1 \tag{17}$$

时它必定出现，因为这些是同时满足(14)和(16)的 $B_1, \cdots, B_{P+1}$ 仅有的值。把(17)值代入(15)，就给出寻道总次数的相当简单的公式

$$(\sqrt{L_1} + \cdots + \sqrt{L_{P+1}})^2/M \tag{18}$$

这可以和数 $(P+1)(L_1 + \cdots + L_{P+1})/M$ 相比较，此数为在所有缓冲区的长度相等时得到的。由习题 1.2.3-31，改进为

$$\sum_{1 \leqslant j < k \leqslant P+1} (\sqrt{L_j} - \sqrt{L_k})^2/M$$

可惜的是，公式(18)并不像定理 K 中那样，易于利用公式(18)确定最优合并型式(参见习题 14)。

链接的使用 M. A. Goetz [*CACM* **6**(1963)，245～248] 提出了一个通过把个别的磁道链接在一起，来避免输出时的寻道时间的有趣方法。他的思想要求一组相当特别的磁盘存储管理程序，而且除了排序之外，它还适用于许多问题。因此对于一般的应用来说，它是一个非常有价值的技术。

它的概念是简单的：代替在磁盘的柱面内顺序地分配磁道，我们把它们链接在一起并且建立一个可用空间的表，每个柱面一个。当要输出一个磁道的信息时，我们把它写到当前的柱面上(无论存取臂在哪里)，除非该柱面满了。这样一来，寻道时间就消失了。

困难之处是在磁道本身我们不能存储对下个磁道的链接，因为所需的信息在当时并不知道(如果合适的话，我们可以存储一个对前一个磁道的链接，而且在下次扫描时向后读文件)。对于每个文件的磁道的链接地址表可以分别地维护，因为它需要比较少的空间。可利用空间表可以通过使用二进位表紧凑地表示，且以 1000 个二进位确定 1000 个磁道是否可用。

修改的预报 算法 5.4.6F 表明，通过考察每个缓冲区中最后的键码，我们可以预报一个 P 路合并的缓冲区哪一个将首先变空。因此我们可以同时读和计算。该算法使用浮动输入缓冲区，它们不是专门提供给一个特定的文件；所以这些缓冲区必须全都具有相同大小，而且不能使用上边介绍的缓冲区分配技术。但是对于一致的缓冲区大小的限制不是大的损失，因为现在的计算机较之以前有大得多的内部存储。现在常常提出一个自然的缓冲区大小，例如，整个磁道的容量作为缓冲区。

因此想像把每个均由一数据块序列组成的 P 个路段合并在一起，其中每个块(或许最后一个除外)，恰包含 B 个记录。D. L. Whitlow 和 A. Sasson 提出了一个叫做 SyncSort[*U.S. Patent 4210961* (1980)]的有趣算法，通过只需要大小为 B 的 3 个

缓冲区和容纳 PB 个记录和 PB 个指针的一个存储池,它改进了算法 5.4.6F。与之相对照,算法 5.4.6F 需要 $2P$ 个输入缓冲区和 2 个输出缓冲区,但不需要指针缓冲区。

把 3 个 SyncSort 缓冲区安排在一个圆周上,当合并进行时,计算机在当前的缓冲区中处理数据,同时把输入读到下一个缓冲区中,并从第三个缓冲区来写输出。许多磁盘设备允许在同一柱面同时读和写。其实,磁盘合并允许我们写新的路段来代替我们正在读的路段。否则我们可以使用两个磁盘,一个用作读,一个用作写。如果完全的读写同步是不可能的(例如,由于寻道时间的原因),我们也可以如 1.4.4 小节中的图 26 所示,把这个圆周扩展成 4 个或者更多的缓冲区。

SyncSort 通过读每个路段的第一个块开始,并把这 PB 个记录放进缓冲区池中。存储池中的每个记录被链接到它所属路段中的它的后继者,但每个块的最后一个记录没有后继者因而除外。在这些最后的记录中键码最小者确定将首先需要填满的路段,所以我们开始读该路段的第二个块。只要第二个块被读入,合并即开始;通过考察它最后的键码,我们就能准确地预报下一个相关的块,而且我们可以以相同的方式预取恰好的块数来输入,而且恰在需要它们之前。

SyncSort 合并算法交换当前缓冲区中的每个记录和输出的下个记录,即存储池中有最小键码的记录。当我们进行每一个交换时,也适当地更新选择树和后继者链接。一旦达到了当前缓冲区的末尾,我们就做好了转动缓冲区圆周的准备:输入缓冲区变成当前缓冲区,写缓冲区用作输入,而且我们开始从以前的当前缓冲区来写。

使用多个磁盘 磁盘设备在大小和重量上曾经非常大,但在 20 世纪 80 年代后,它们明显变得越来越小、越来越轻和越来越廉价——而且比以前任何时候都能容纳更多的数据。因此,人们开始设计用于 5 个、10 个或 50 个磁盘设备的、曾经是不可想像的磁盘丛组,甚至更大的磁盘群的算法。

对于附加的磁盘要想提高速度一个容易的方法是对大的文件使用*磁盘分片*(disk striping)技术。假设我们有 D 个磁盘机,分别编号为 $0,1,\cdots,D-1$,另外一个文件由 L 个块 $a_0a_1\cdots a_{L-1}$ 组成。把这个文件在 D 个磁盘上分片意味着把块 a_j 放在编号为 $j \bmod D$ 的磁盘上;于是磁盘 0 容纳 $a_0a_Da_{2D}\cdots$,磁盘 1 容纳 $a_1a_{D+1}a_{2D+1}\cdots$,等等。于是在 D 块群 $a_0a_1\cdots a_{D-1}$, $a_Da_{D+1}\cdots a_{2D-1}$, $\cdots$,我们可以同时实现 D 个读或写,这些块群称做*超块*(superblock)。每个超块中的独立的块应在不同磁盘上的对应柱面上,使得每个磁盘机的寻道时间是相同的。实际上,我们所做的就好像我们有一个磁盘,它具有大小为 DB 的块和缓冲区,不同的是,输入和输出操作都快了 D 倍。

当我们进行两路合并时,或者一般地,每当我们要匹配以键码为序的两个文件中有相同键码的记录时,可以利用对于超块分片的一个很好的改进。假设第一个文件的块 $a_0a_1a_2\cdots$ 像上面所述被分片到 D 个磁盘上,而另一个文件的块 $b_0b_1b_2\cdots$ 以相反的次序来分片,且块 b_j 在编号为 $(D-1-j) \bmod D$ 的磁盘上。例如 $D=5$,则块 a_j 分别出现在 $0,1,2,3,4,0,1,\cdots$ 的磁盘上,而对于 $j\geqslant 0$,块 b_j 出现在 $4,3,2,1,0,4,3,\cdots$ 的磁盘上。令 α_j 是块 a_j 最后的键码,而设 β_j 是块 b_j 的最后的键码。通过

考察 α 和β,我们可以预报我们将要读数据块的序列。例如,这个序列可能是

$$a_0b_0a_1a_2b_1\quad a_3a_4b_2a_5a_6\quad a_7a_8b_3b_4b_5\quad b_6b_7b_8b_9b_{10}\quad \cdots$$

当 $D=5$ 时,这些块分别出现在磁盘

$$0\ 4\ 1\ 2\ 3\quad 3\ 4\ 2\ 0\ 1\quad 2\ 3\ 1\ 0\ 4\quad 3\ 2\ 1\ 0\ 4\quad \cdots$$

上,而且如果一次读其中的 5 个,则我们将逐次地从磁盘{0,4,1,2,3},{3,4,2,0,1},{2,3,1,0,4},{3,2,1,0,4}…进行输入。这里绝不会有冲突出现,即绝不会在同一时间里要从同一个磁盘读两个记录!一般地说,使用 D 个磁盘,我们可以无冲突地一次读 D 个块,因为对于某个 k,第一个群将有 k 个块在 0 到 $k-1$ 的磁盘上,以及有 $D-k$ 个块 $b_0\cdots b_{D-k-1}$ 在 $D-1$ 到 k 的磁盘上;因此我们将以同样的方式平稳地继续,但磁盘的编号由 k 来循环地移动。

这个技巧是扑克牌魔术师们所熟知的,他们把它称做 Gilbreath 原理;它是 20 世纪 60 年代由 Norman Gilbreath[参见 Martin Gardner, *Mathematical Magic Show* (New York: Knopf, 1977), Chapter 7; N. Gilbreath, *Genii* **52**(1989), 743～744]发明的。我们需要知道 α 和β,来判定下次应读什么块,由于该信息仅占据 a 和 b 所需要的空间的一个很小比例,因此它可以被保存在独立的文件中。这样,为使输入全速进行,我们只需要较少的缓冲区即可(参见习题 23)。

随机分片 当 P 和D 很大时,如果我们要用 D 个磁盘来进行 P 路合并,我们不可能同时从 D 个磁盘读信息而保持不冲突,除非我们有大量的缓冲区,因为当 $P>2$ 时,我们没有和 Gilbreath 原理相类似的东西。不管我们怎样分配一个文件的块到磁盘上去,都将会有这样的机会,就是在我们准备好使用许多块之前,可能需要把它们读入内存中,因为真正需要的这些块可能会碰巧驻留在相同的磁盘上。

例如,假设我们要在 5 个磁盘上进行八路合并,并且假设 8 个路段的块 $a_0a_1a_2\cdots,b_0b_1b_2\cdots,\cdots,h_0h_1h_2\cdots$ 已经通过在 $j \bmod D$ 磁盘上的 a_j,$(j+1)\bmod D$ 磁盘上的 b_j,…,$(j+7)\bmod D$ 磁盘上的 h_j 来分片。我们可能需要以下列次序来访问这些块

$$a_0b_0c_0d_0e_0\ f_0g_0h_0d_1e_1\ d_2e_2d_3a_1f_1\ b_1g_1a_2f_2e_3\ d_4c_1h_1b_2g_2\ a_3f_3e_4d_5d_6\cdots \tag{19}$$

于是它们就分别出现在下列磁盘上

$$0\,1\,2\,3\,4\,0\,1\,2\,4\,0\,0\,1\,1\,1\,1\,2\,2\,2\,2\,2\,2\,3\,3\,3\,3\,3\,3\,3\,3\,4\cdots \tag{20}$$

所以我们最好的办法是如下来输入它们

时间 1	时间 2	时间 3	时间 4	时间 5
$a_0b_0c_0d_0e_0$	$f_0g_0h_0c_1d_1$	$e_1e_2b_1h_1d_6$	$d_2d_3g_1b_2?$	$?a_1a_2g_2?$
时间 6	时间 7	时间 8	时间 9	
$?f_1f_2a_3?$	$??e_3f_3?$	$??d_4e_4?$	$???d_5?$	

(21)

在我们有能力来考察块 d_5 时,我们需要已经读 d_6 以及由"?"所标记的 15 个未来数据的块,这是由于在磁盘 3 上的拥挤所致。而且,通过包含 a_3,b_2,c_1,e_4,f_3,g_2 和 b_1 的剩余部分的 7 个缓冲区,我们还不能完成;所以在这个特定的例子里,我们还将

需要保存至少(16 + 8 + 5)B 个输入记录的缓冲区空间。

磁盘分片的简单超块方法将以下列方式进行，即：在时间 1 读块 $a_0a_1a_2a_3a_4$，在时间 2 读 $b_0b_1b_2b_3b_4$，…，在时间 8 读 $h_0h_1h_2h_3h_4$，然后在时间 9 读 $d_5d_6d_7d_8d_9$（因为 $d_5d_6d_7d_8d_9$ 为下次需要的超块），等等。使用 SyncSort 的策略，在内存中将需要保存$(P+3)DB$ 个记录的缓冲区和 PDB 指针。上面所述的更加多样性的方法可被证明仅需要大约一半的缓冲空间，但当 P 和 D 很大时，内存要求仍然近似于和 PDB 成比例（参见习题 24）。

R. D. Barve, E. F. Grove 以及 J. S. Vitter[Parallel Computing **23**(1997), 601～631] 证明，对于独立块方法的一个很小的修改将导致这样一个算法，它使磁盘的输入/输出运行在接近它的全速之下，同时只需要 $O(P + D\log D)$ 个缓冲块而不是 $\Omega(PD)$。他们的随机分片技术把路段 k 的块 j 放进$(x_k + j) \bmod D$ 磁盘上，其中 x_k 是在路段 k 首先被写出之前刚刚被选择的一个随机整数。取代替总是不变地输入 D 个块，从每一个磁盘输入一个，他们提出，当没有足够的空间来保持在某些磁盘上向前读时，阻止输入的一个简单机制，而且他们证明了，他们的方法是接近最优的。

为了使用随机分片在 D 个磁盘上进行 P 路合并，我们可以维护 $2D + P + Q - 1$ 个浮动的输入缓冲区，每一个容纳 B 个记录的一个块。输入典型地读到这些缓冲区中的 D 个，我们称其为*活动的读缓冲区*，另外 P 个缓冲区包含其中记录当前正被合并的前一些块，这些缓冲区称做*活动的合并缓冲区*。其余的 $D + Q - 1$ 个"空白"缓冲区或者是空的，或者保存预先取得的数据，稍后将需要它们。Q 是一个非负参数，它可以被增值以便来减少在任何盘上输入被阻止的机会。

所有路段的块如(19)中那样，被安排为以时间为序：首先我们列出每个路段的块 0，然后通过确定活动的合并缓冲区变为空的顺序，列出其它块。像上面所说明的那样，这个次序是由每个块中最后的键码确定的，所以我们很容易地预报应该预先取得哪些块。

对 $P = 8, D = 5$ 和 $Q = 4$ 再次考虑例(19)。现在我们仅有 $2D + P + Q - 1 = 21$ 个输入缓冲块，而不是上面所述为了极大速度读入所需要的 29 个。我们将使用下列偏离（由 π 的十进数字所提示的）

$$x_1 = 3,\ x_2 = 1,\ x_3 = 4,\ x_4 = 1,\ x_5 = 0,\ x_6 = 4,\ x_7 = 2,\ x_8 = 1 \quad (22)$$

作为路段 $a, b, \cdots, h$；这样，如果我们以时间为序列出它们的块的话，磁盘分别包含

磁盘	块
0:	e_0　f_1　a_2　d_4c_1　…
1:	b_0　d_0　h_0　e_1　f_2　a_3　d_5　…
2:	g_0　d_1　e_2　b_1　h_1　f_3　d_6　…
3:	a_0　d_2　g_1　e_3　b_2　…
4:	c_0　f_0　d_3a_1　g_2　e_4　…

(23)

(22)的"随机"偏离，连同每一个路段内的顺序分片一起，将趋向于极小化任何特定的以时间为序所产生的拥挤。实际的处理类似下边这样进行：

	活动读	活动合并	空白	等候	
时间 1	$e_0b_0g_0a_0c_0$	– – – – – – – – –	(– – – – – – – – –)	a_0	
时间 2	$f_1d_0d_1d_2f_0$	a_0 – – – – – – – –	$b_0c_0(e_0g_0$ – – – –)	d_0	
时间 3	$a_2h_0e_2g_1d_3$	$a_0b_0c_0d_0$ – – – –	$e_0f_0g_0(d_1d_2f_1$ – –)	h_0	
时间 4	$a_2e_1b_1g_1a_1$	$a_0b_0c_0d_0e_0f_0g_0h_0$	$d_1(d_2e_2d_3f_1g_1a_2$ –)	e_1	(24)
时间 5	$d_4f_2h_1e_3g_2$	$a_0b_0c_0d_1e_1f_0g_0h_0$	$d_2e_2d_3a_1f_1b_1g_1a_2()$	f_2	
时间 6	$c_1a_3f_3b_2e_4$	$a_2b_1c_0d_3e_2f_2g_1h_0$	$e_3d_4(h_1g_2$ – – – –)	c_1	
时间 7	? d_5d_6? ?	$a_2b_1c_1d_4e_3f_2g_1h_0$	$h_1b_2g_2a_3f_3e_4$(– –)	d_5	

在每一个时间单位,我们等候还未被合并并且也未在一个空缓冲区中的按时间来说的第一个块;这是当前被输入到一个活动读缓冲区的块之一。我们假定,计算机比磁盘要快得多;因此,在输入完成之前,在我们等候的那个块之前的所有块都将已进入合并过程。我们还假定,有充分的输出缓冲区可利用,使得合并不会由于缺乏位置来放置输出而被延迟(参见习题 26)。当一轮输入完成时,我们等候的那个块立即被分类为活动合并缓冲区,而且它原占据的空的合并缓冲区将被用作下一个活动的读缓冲区。其它 $D-1$ 个活动的读缓冲区现在和 $D-1$ 个最不重要的空白缓冲区交换;空白缓冲区是按它们内容的时间顺序来排序的。在下一轮,我们将等候不在空白缓冲区中的第一个未合并的块。在时间次序下居于该块之前的任何空白缓冲区,在下一输入循环之前将变成活动合并的一部分。但是其它——上面括号内所示部分——将继续,而且在下一轮仍将保留作为空白缓冲。但是,在括号内的缓冲区中至多有 Q 个可以继续,因为在输入就绪之后,我们需立即把 $D-1$ 个空白缓冲区转换成活动读的状态。任何另外的空白缓冲区将有效地被腾出,就如同它们还未被读入那样。此腾出出现在(24)中的时间 4,我们不能把所有 6 个块 $d_2e_2d_3f_1g_1a_2$ 继续到时间 5,因为 $Q=4$,所以我们重读 g_1 和 a_2。否则这个例子中的读操作就以全速进行。

给定有待合并路段的任何时间顺序,习题 29 证明,随机分片方法平均说来,在 $r(D,Q+2)$的一个因式内,将实现极小个数的磁盘读,其中函数 r 列于表 2 中。例如,如果 $D=4$ 和 $Q=18$,则用 4 个磁盘和 $P+25$ 个输入缓冲对 L 个数据块进行 P 路合并的平均时间,将至多是在一个磁盘上读 $r(4,20)L/D\approx1.785L/4$ 个块的时间。这个理论的上限是十分保守的;实践中,性能是更好的,它非常接近于最优时间 $L/4$。

表 2 随机分片性能的保证

	$r(d,d)$	$r(d,2d)$	$r(d,3d)$	$r(d,4d)$	$r(d,5d)$	$r(d,6d)$	$r(d,7d)$	$r(d,8d)$	$r(d,9d)$	$r(d,10d)$
$d=2$	1.500	1.500	1.499	1.467	1.444	1.422	1.393	1.370	1.353	1.339
$d=4$	2.460	2.190	1.986	1.888	1.785	1.724	1.683	1.633	1.597	1.570
$d=8$	3.328	2.698	2.365	2.183	2.056	1.969	1.889	1.836	1.787	1.743
$d=16$	4.087	3.103	2.662	2.434	2.277	2.156	2.067	1.997	1.933	1.890
$d=32$	4.503	3.392	2.917	2.654	2.458	2.319	2.218	2.130	2.062	2.005

（续）

	r(d,d)	r(d,2d)	r(d,3d)	r(d,4d)	r(d,5d)	r(d,6d)	r(d,7d)	r(d,8d)	r(d,9d)	r(d,10d)
d=64	5.175	3.718	3.165	2.847	2.613	2.465	2.346	2.249	2.174	2.107
d=128	5.431	3.972	3.356	2.992	2.759	2.603	2.459	2.358	2.273	2.201
d=256	5.909	4.222	3.536	3.155	2.910	2.714	2.567	2.464	2.363	2.289
d=512	6.278	4.455	3.747	3.316	3.024	2.820	2.675	2.556	2.450	2.375
d=1024	6.567	4.689	3.879	3.434	3.142	2.937	2.780	2.639	2.536	2.452

键码排序有效吗？ 当记录均很长而键码均很短时，建立只由键码以及确定它们原始文件位置的一系列的数组成的新文件，是特别有吸引力的。在对这个键码文件排序之后，我们可以通过连续的数 1,2,…来代替这些键码；这样新文件就可以根据原始文件的位置排序，而且我们可以方便地说明如何整理和最后重新安排记录。这个过程可表示如下：

i)原始的文件	$(K_1,I_1)(K_2,I_2)\cdots(K_N,I_N)$	长
ii)键码文件	$(K_1,1)(K_2,2)\cdots(K_N,N)$	短
iii)排好序的 ii)	$(K_{p_1},p_1)(K_{p_2},p_2)\cdots(K_{p_N},p_N)$	短
iv)编辑后的 iii)	$(1,p_1)(2,p_2)\cdots(N,p_N)$	短
v)排好序的 iv)	$(q_1,1)(q_2,2)\cdots(q_N,N)$	短
vi)编辑后的 i)	$(q_1,I_1)(q_2,I_2)\cdots(q_N,I_N)$	长

这里 $p_j=k$ 当且仅当 $q_k=j$。iii)和 v)中的两个排序过程是比较快的(甚至可能是内部排序)，因为记录都不很长。在阶段 vi)中，我们已经把这个问题归结为对其键码只不过是数$\{1,2,\cdots,N\}$的一个文件进行排序的问题；每个记录现在精确地指明它要被移到何处。

乍一看去，阶段 vi)之后剩下的外部重排问题似乎是简单的。但事实上，它是相当困难的，而且还并未找到真正好(比排序好得多)的算法。显然我们可以在 N 步之内进行重排，一次移动一个记录。对于足够大的 N，这要比一个排序方法的 $N\log N$ 更好些。但是 N 绝不会这么大，然而 N 也还是非常大的，因为 N 个寻道是不可想像的。

对于 vi)的编辑后的记录可以有效地使用基数排序方法，因为它们的键码有一个完全一致的分布。在现代计算机上，八路分布的处理时间比起八路合并的处理时间要快得多；因此分布排序大概是最好的过程(参见 5.4.7 小节，也可参见习题 19)。

另一方面，在键码已经排好序之后，来做一个完全的排序似乎是浪费。一个原因是，外部重排问题意料不到地困难。这点已由 R.W.Floyd 所发现，他找到了为在一个磁盘设备上重排记录所需要的寻道次数的一个重要下限[*Complexity of Computer Computations* (New York:Plenum,1972),105～109]。

借助于 5.4.8 小节中的电梯问题，来描述 Floyd 的结果是方便的；但这次我们要来寻找一个极小化停止次数，而不是运行距离的电梯调度。使停止次数极小并不完全等同于极小-寻道的重排算法，因为每次停止都把对电梯的输入和从电梯的输

出结合起来;但是停止极小化准则十分接近于这里的基本思想。

我们将利用“离散熵”(discrete entropy)函数

$$F(n)=\sum_{1<k\leqslant n}(\lceil \lg k\rceil+1)=B(n)+n-1=n\lceil \lg n\rceil-2^{\lceil \lg n\rceil}+n \qquad (25)$$

其中 $B(n)$是二叉插入函数,即等式 5.3.1-(3)。由等式 5.3.1-(34),$F(n)$是有 n 个叶的一个二叉树的极小外部通路长度,且

$$n\lg n\leqslant F(n)\leqslant n\lg n+0.0861n \qquad (26)$$

由于 $F(n)$是凸的且满足 $F(n)=n+F(\lfloor n/2\rfloor)+F(\lceil n/2\rceil)$,由上面的引理 C 我们知道

$$F(n)\leqslant F(k)+F(n-k)+n \qquad \text{对于 } 0\leqslant k\leqslant n \qquad (27)$$

从 F 的外部通路长度的特征来看,这个关系也是明显的;这个关键性的事实下面还要用到。

像在 5.4.8 小节中一样,我们假定,每一层可容纳 b 个人,电梯可容纳 m 个人,此楼共有 n 层。令 S_{ij} 为当前在 i 层而其目的地是 j 层的人数。在这个建筑物中人员的任何配置的集中率(togetherness rating)定义为和 $\sum_{1\leqslant i,j\leqslant n}F(S_{ij})$。

例如,假定 $b=m=n=6$,而且开始时,36 个人的分布如下:

```
⊔⊔⊔⊔⊔⊔
123456    123456    123456    123456    123456    123456        (28)
```

电梯是空的,停在 1 层上;“⊔”表示一个空位置。在每一层上,对每种可能的目的地,均有一个人与之相应,因此所有的 s_{ij} 都为 1,且集中率为 0。如果现在电梯运载 6 个人到第 2 层,我们有下列的配置

```
              123456
⊔⊔⊔⊔⊔⊔        123456    123456    123456    123456    123456    (29)
```

而集中率变为 $6F(0)+24F(1)+6F(2)=12$。现在假设电梯运载 1,1,2,3,3 和 4 到 3 层:

```
                        112334
⊔⊔⊔⊔⊔⊔        245566    123456    123456    123456    123456    (30)
```

则集中率跃升为 $4F(2)+2F(3)=18$。当所有的人都被运载到他们的目的地时,集中率将是 $6F(6)=96$。

Floyd 发现,在每次停止之后,集中率的增加绝不会多于 $b+m$,因为一组 s 个相同目的地的人同一组 s' 个类似 $F(s+s')-F(s)-F(s')\leqslant s+s'$。因此我们有下列的结果。

定理 F *令 t 为上述定义下,bn 个人初始配置的集中率。为把他们全部运到目的地,电梯至少必须停*

$$\lceil (F(b)n-t)/(b+m)\rceil$$

次。 ▌

把这个结果用磁盘术语表达,即:假设有 bn 个记录,且每个块有 b 个记录,内存一次可容纳 m 个记录。每一次磁盘读都读一个块到内存中,每一次磁盘写则存储

一个块，s_{ij}是块 i 中属于块 j 的记录的个数。如果 $n \geqslant b$，则存在这样的初始配置，其中所有 $s_{ij} \leqslant 1$；所以 $t=0$，且为了重排这些记录，至少需要 $f(b)n/(b+m) \approx (bn\lg b)/m$ 个读块操作（当 b 很大时，因子 $\lg b$ 使这个下限不可忽略）。习题 17 对于 m 比 b 大得多的一般情况，导出了一个强得多的下限。

习 题

1.［*M22*］正文中介绍了一个方法，通过它，为读一个磁道的一部分 x 所需要的平均等待时间从$\frac{1}{2}$减少到$\frac{1}{2}(1-x^2)$圈转动。当有一个存储臂时，这是极小的值。如果有两个存取臂，相距 180°，任一时刻仅有一个臂可传送数据，则对应的极小平均等待时间是什么。

2.［*M30*］（A.G.Konheim）本问题的目的是来考察，当合并被垂直分配于柱面的文件时，一个磁盘的存取臂必须移动多远。假设有 P 个文件，每个含 L 个记录块，并且假定每个文件的第一个块出现在柱面 1 上，第 2 个块出现在柱面 2 上，等等。在每个块中最后的键码的相对次序在合并期间支配存取臂的运动，因此我们可以以下列在数学上可处理的方式表示这一状况：考虑 PL 个有序对的一个集合

$$\begin{matrix} (a_{11},1) & (a_{21},1) & \cdots & (a_{P1},1) \\ (a_{12},2) & (a_{22},2) & \cdots & (a_{P2},2) \\ \vdots & \vdots & & \vdots \\ (a_{1L},L) & (a_{2L},L) & \cdots & (a_{PL},L) \end{matrix}$$

其中集合$\{a_{ij} \mid 1 \leqslant i \leqslant P, 1 \leqslant j \leqslant L\}$由在某种顺序下的数$\{1,2,\cdots,PL\}$组成，且其中对于 $1 \leqslant i < L$，$a_{ij} < a_{i(j+1)}$（行表示柱面，列表示输入文件），以它们的第一个分量来对这些对排序并令得到的序列为$(1,j_1)(2,j_2)\cdots(PL,j_{PL})$。试证明，如果 a_{ij}的$(PL)!/L!^P$ 个选择的每一个都是同等可能的，则

$$|j_2-j_1|+|j_3-j_2|+\cdots+|j_{PL}-j_{PL-1}|$$

的平均值是

$$(L-1)\left(1+(P-1)2^{2L-2}\Big/\binom{2L}{L}\right)$$

提示：参见习题 5.2.1-14。注意当 $L\to\infty$时这个值渐近等于$\frac{1}{4}(P-1)L\sqrt{\pi L}+O(PL)$。

3.［*M15*］假设内存有限使得十路合并是不可行的。问如何修改递推关系(3)，(4)，(5)，使得在所有的具有 n 片叶子且没有叉数大于 9 的内部节点的树$\mathcal{T}$中，$A_1(n)$是 $\alpha D(\mathcal{T})+\beta E(\mathcal{T})$的极小值？

▶4.［*M21*］考虑平方根缓冲区分配方案的一个修改形式，其中所有 P 个输入缓冲区有相同的长度，但是应当选择输出缓冲区以便极小化寻道时间。

a)对于一 L 个字符的 P 路合并，推导出对应于(2)的一个运行时间公式。

b)试证明，可以修改定理 K 中的构造，以得到这样一个合并型式，按照你从 a)中得到的公式，这个型式是最优的。

5.［*M20*］当使用两个磁盘，以便在一个上的读入和在另外一个上的写相重叠时，我们不可能使用如图 93 所示的合并型式，因为某些叶在偶数级上而某些叶在奇数级上。说明如何修改定理 K，以便产生出这样的树，即在所有的叶都出现在偶数级上或都出现在奇数级上这一约束条件

下，它们是最优的。

▶6.[*22*] 当 $n=23$ 且 $\alpha=\beta=1$ 时，求在习题 5 的意义下最优的一株树（你可能希望使用计算机来做）。

▶7.[*M24*] 当初始路段不全有相同长度时（在定理 H 的意义下），最好的合并型式使 $\alpha D(\mathcal{T})+\beta E(\mathcal{T})$ 极小，其中 $D(\mathcal{T})$ 和 $E(\mathcal{T})$ 此处表示加权的路径长度：把权 $w_1,\cdots,w_n$（对应于初始路段的长度）附加到树的每个叶，而且叉数之和与路径长度乘以适当的权。例如，如果 $\mathcal{T}$ 是图 92 的一株树，则我们将有 $D(\mathcal{T})=6w_1+6w_2+7w_3+9w_4+9w_5+7w_3+4w_7+4w_8$，$E(\mathcal{T})=2w_1+2w_2+2w_3+3w_4+3w_5+2w_6+w_7+w_8$。

试证明，总有一个最优树，其中，对于某个 k，首先合并最短的 k 个路段。

8.[*49*] 在习题 7 的意义下，是否有一个算法，对于给定的 α,β 和权 $w_1,\cdots,w_n$，它对某个 c，仅用 $O(n^c)$ 步，就找出最优树？

9.[*HM39*] (L. Hyafil, F. Prusker, J. Vuillemin) 证明，对于固定的 α 和 β，当 $n\to\infty$ 时

$$A_1(n)=\left(\min_{m\geqslant 2}\frac{\alpha m+\beta}{\log m}\right)n\log n+O(n)$$

其中 $O(n)$ 项 $\geqslant 0$。

10.[*HM44*] (L. Hyafil, F. Prusker, J. Vuillemin) 证明，当 α 和 β 固定时，如果 m 使习题 9 中的系数极小，则对于所有充分大的 n，$A_1(n)=\alpha mn+\beta n+A_m(n)$。

11.[*M29*] 使用习题(6)和(11)的符号，证明对于所有的 $m\geqslant 2$ 和 $n\geqslant 2$，$f_m(n)+mn\geqslant f(n)$，并确定使等式成立的所有 m 和 n。

12.[*25*] 试证明，对所有 $n>0$，存在具有 n 个叶和极小叉数路径长度(6)的一株树，其所有的叶都在同一级上。

13.[*M24*] 试证明，对于 $2\leqslant n\leqslant d(\alpha,\beta)$，其中 $d(\alpha,\beta)$ 在(12)中定义，在定理 H 的意义下惟一最好的合并型式是一个 n 路合并。

14.[*40*] 若使用缓冲区分配的平方根方法，则对图 92 中的合并型式的寻道时间将与 $(\sqrt{2}+\sqrt{4}+\sqrt{1}+\sqrt{1}+\sqrt{8})^2+(\sqrt{1}+\sqrt{1}+\sqrt{2})^2+(\sqrt{1}+\sqrt{2}+\sqrt{1}+\sqrt{4})^2+(\sqrt{1}+\sqrt{1}+\sqrt{2})^2$ 成正比；这实际上是每个内部节点的 $(\sqrt{n_1}+\cdots+\sqrt{n_m}+\sqrt{n_1+\cdots+n_m})^2$ 的和，其中各节点对应的子树有 $(n_1,\cdots,n_m)$ 个叶。基于这个公式，试编写一个计算机程序，它生成有 1,2,3,…个叶的极小寻道时间树。

15.[*M22*] 试证明，如果电梯开始时为空，且如果 $F(b)n\neq t$，则可对定理 F 稍做改进，即：在这样的情况下，至少 $\lceil F(b)n+m-t)/(b+m)\rceil$ 次停止是必须的。

16.[*23*] (R. W. Floyd) 试求一个电梯调度，它运载(28)中的所有人到他们的目的地最多停 12 次（配置(29)表明一次停止，而不是两次之后的状况）。

▶17.[*HM25*] (R. W. Floyd, 1980) 由于某些初始配置必定要求下述这么多次的停止，试证明定理 F 的下限可改进为

$$\frac{n(b\ln n-\ln b-1)}{\ln n+b(1+\ln(1+m/b))}$$

提示：计算在 s 次停止之后可以排序的配置数。

18.[*HM26*] 设 L 是习题 17 的下限。试证明，为把所有人带到他们的目的楼层所需要的电梯停止的平均次数，当人们进入到 bn 个磁盘的所有 $(bn)!$ 种可能的排列都是同等可能的时，至少为 $L-1$。

▶19.[*25*] (B. T. Bennett 和 A. C. Mckellar) 考虑下列键码排序的方法（以含 10 个键码的一个示例文件来说明）：

i)源文件：$(50,I_0)(08,I_1)(51,I_2)(06,I_3)(90,I_4)(17,I_5)(89,I_6)(27,I_7)(65,I_8)(42,I_9)$

ii)键码文件：(50,0)(08,1)(51,2)(06,3)(90,4)(17,5)(89,6)(27,7)(65,8)(42,9)

iii)排序后的 ii):(06,3)(08,1)(17,5)(27,7)(42,9)(50,0)(51,2)(65,8)(89,6)(90,4)

iv)箱指定(见下面):(2,1)(2,3)(2,5)(2,7)(2,8)(2,9)(1,0)(1,2)(1,4)(1,6)

v)排序后的 iv):(1,0)(2,1)(1,2)(2,3)(1,4)(2,5)(1,6)(2,7)(2,8)(2,9)

vi)使用 v)把 i)分布到箱中:

箱 1:$(50,I_0)(51,I_2)(90,I_4)(89,I_6)$

箱 2:$(08,I_1)(06,I_3)(17,I_5)(27,I_7)(65,I_8)(42,I_9)$

vii)替代选择的结果,首先读箱 2,然后读箱 1:

$(06,I_3)(08,I_1)(17,I_5)(27,I_7)(42,I_9)(50,I_0)(51,I_2)(65,I_8)(89,I_6)(90,I_4)$

步骤 iv)中箱号的指定,是通过对 iii)做替代选择,*从右到左*,且以第二个分量的*递减*次序来进行。箱号即路段号。上面的例子使用在选择树中仅有两个元素的替代选择。在 iv)和 vii)中对替代选择应使用相同大小的树。注意箱的内容不必是排序的!

试证明,这个方法将完成排序,即 vii)中替代选择将仅产生一个路段(这个技术通过分布而减少在传统的键码排序中所需要的箱子数,特别是当输入大部分已处于有序时)。

▶20.[*25*] 现代的硬件/软件系统为程序员提供了一个*虚拟内存*,使得编写程序时,就好像有一个非常大的、可以包含所有数据的内存一样。此内存分为一些页,在某一个时刻只有一小部分在真正的内存当中,其它的则在磁盘或磁鼓上。程序员不必关心这样的细节,因为系统会处理每一件事;当需要时新的页会被自动地放进内存中。

看起来,虚拟内存技术的发明使外部排序方法显得有些过时,因为使用针对内部排序所开发的技术就可简单地完成工作。试讨论这一情况,问在什么方式下,一个定制的外部排序方法要胜过一个应用通用分页技术的内部排序方法?

▶21.[*M15*] 当把文件分片到 D 个磁盘时,一个 L 个块的文件中有多少块会分到磁盘 j 上?

22.[*22*] 如果使用 Gilbreath 原理合并两个文件,而且要以 a 个块来存键码 α_j,以 b 个块来存键码 β_j,问为在需要时即有信息可用,α_j 应被放置在哪个块中?

▶23.[*20*] 当分别以(a)超块分片和(b)Gilbreath 原理来进行两路合并时,为使输入连续地进行,输入缓冲区需要多少空间?

24.[*M36*] 假设已经把 P 个路段分片到 D 个磁盘上,使得路段 k 的块 j 出现在磁盘 $(x_k+j) \bmod D$ 上。一个 P 路合并将以如(19)那样的某个时间顺序读这些块。如果要连续地输入多组 D 个块,则如(21)中那样,我们将在时刻 t 读每个磁盘上所存的第 t 个块。无论时间顺序如何,为保存还未被合并的数据,内存中需要的缓冲存记录的极小个数是多少?说明如何选择偏离值 x_1, $x_2,\cdots x_p$,使得在最坏情况下需要的缓存最少。

25.[*23*] 对于 $Q=3$ 而不是 $Q=4$ 的情况,重做正文中随机分片的示例。与(24)不同,什么缓冲区的内容将出现?

26.[*26*] 有多少输出缓冲区能够保证,随机化分片的 P 路合并不会由于内存中缺乏位置来放置新近合并的输出而停止?假定写一个块的时间等于读一个块的时间。

27.[*HM27*](*循环占有问题*) 假设把 n 个空瓶子放成一个圆并且指定了编号 $0,1,\cdots,n-1$。对于 $k=1,2,\cdots,p$,我们把 m_k 球掷到瓶 $(X_k+j) \bmod n$ 中,其中 $j=0,1,\cdots,m_k-1$,且整数 x_k 是随机选择的。设 $S_n(m_1,\cdots,m_p)$ 是在瓶 0 中的球数,设 $E_n(m_1,\cdots,m_p)$ 是在最满的瓶中预期的球数。

a)证明 $E_n(m_1,\cdots,m_p) \leqslant \sum_{t=1}^{m} \min(1, n\Pr(S_n(m_1,\cdots,m_p) \geqslant t))$,其中 $m=m_1+\cdots+m_p$。

b)使用尾部不等式,等式 1.2.10-(25),证明对于任何非负实数 $\alpha_1,\alpha_2,\cdots,\alpha_m$

$$E_n(m_1,\cdots,m_p) \leqslant \sum_{t=1}^{m} \min\left(1, \frac{n(1+\alpha_t/n)^m}{(1+\alpha_t)^t}\right)$$

$\alpha_1, \cdots, \alpha_m$ 的什么值给出最好的上限?

28.[*HM47*] 继续 27 题,$E_n(m_1, \cdots, m_p) \geqslant E_n(m_1 + m_2, m_3, \cdots, m_p)$吗?

▶29.[*M30*] 本题的目的是导出,当块表示 P 个路段和 D 个磁盘时,通过随机分片过程,为输入以时间为序的任何块序列所需要的平均时间的一个上限。我们说,当算法进行时(参见(24)),在每一时间步中所等候的块被"标记";因此总的输入时间和被标记的块个数成正比。标记仅依赖于磁盘存取的时间序列(参见(20))。

a)证明,如果 $Q+1$ 个时间顺序下连续的块中有 N_j 个在磁盘 j 上,则在这些块中至多有 $\max(N_0, N_1, \cdots, N_{D-1})$是被标记的。

b)通过证明对于 $Q+2$ 个连续的块它也成立来加强 a)的结果。

c)给定任何时间顺序,借助于表 2 中的函数 $r(D, Q+2)$,使用习题 27 的循环占有问题,得到平均运行时间的一个上限。

30.[*HM30*] 试证明,当 $s \to \infty$ 时,对于固定的 d,习题 29 的函数 $r(d, m)$满足$r(d, sd\log d) = 1 + O(1/\sqrt{s})$。

31.[*HM48*] 作为 P, Q 和 D 的函数,分析随机分片以确定它真正的平均特性,而不仅仅是一个上限(甚至在 $Q=0$ 的情况下,它需要平均 $\theta(L/\sqrt{D})$的读循环,这是很有趣的)。

5.5 小结、历史和文献目录

既然我们已经接近这极为冗长的一章的结尾,我们最好"整理出"已经研究过的最为重要的事实。

用于排序的一个算法,是一个这样的过程,它重新安排一个文件的所有记录,使得其键码处于递增的次序。这有序的排列是有用的,因为它把相同键码的记录放到一起,允许有效地处理按同一键码排好序的多个文件,这导致了有效的检索算法,而且使计算机的输出看上去不那么混乱。

当所有的记录都置于计算机的高速内存中时,就使用*内部排序*。我们对于内部排序已经研究了很多算法,其详细程度各不相同;不过或许我们并不知道对这个问题有这样多种不同方法时,我们反倒会轻松些! 学习所有这些技术是有趣的,但是现在我们却不得不面临这么一种情况,即在一给定的情况下,到底应该使用哪一种方法。

如果不论是对哪一种应用,或不论正在使用的是什么计算机,仅仅有一两种排序方法,它比所有其它的排序方法都好,那么,事情倒很好解决。但是事实上,每种方法都有它各自的特点。例如,冒泡排序(算法 5.2.2B)并没有明显可取的特性,因为总有一个更好的方法来完成它所能完成的;但即使是这样一项技术,在做适当的拓展之后,证明对于两条磁带的排序是有效的(参见 5.4.8 小节)。因此我们发现,几乎所有的算法都值得去记住,因为在某些特定应用中,它们是最好的。

以下的简短总结,给出了对于内部排序,我们所涉及到的最有意义算法的概述。和通常一样,N 表示给定文件中的记录数。

1. *分布计数*,算法 5.2D 当键码有一个小的变化范围时,此方法非常有用。它是稳定的(不影响具有相同键码记录的次序),但需要存储计数器和 $2N$ 个记录的存

储空间。习题 5.2-13 中给出了在损失稳定性的代价下，节省 N 个记录空间的一个修改。

2. **直接插入，**算法 5.2.1S　对于编程来说是最简单的方法，不需要额外的空间，而且对于较小的 N(比如说 $N\leqslant25$)十分有效。对于大的 N，它慢得不能容忍，除非输入是接近于有序的。

3. **Shell 排序，**算法 5.2.1D　也十分容易编程，而且使用极小的内存空间；它对于大小适当的 N(比如说 $N\leqslant1000$)相当有效。

4. **表插入，**算法 5.2.1L　使用和直接插入相同的基本思想，所以它仅适合于小的 N。如同以下所述的其它表排序方法一样，它通过处理链接减少了移动长记录的花销；当记录的长度可变，或者是其它数据结构的一部分时，特别有效。

5. **地址计算**　当键码有一个已知的(通常是一致的)分布时很有效。这个方法主要的变种是*多表插入*(程序 5.2.1M)，以及 MacLaren 的组合基数插入方法(在 5.2.5 小节的结尾处讨论)，后者仅用 $O(\sqrt{N})$ 个额外内存单元来完成。在定理 5.2.5T 中讨论了两遍扫描的方法。

6. **合并交换，**算法 5.2.2M(Batcher 方法)　此算法和它的"姊妹"算法*双调排序*(习题 5.3.4-10)，在大量的比较可同时进行时很有用。

7. **快速排序，**算法 5.2.2Q(Hoare 方法)　此算法大概是内部排序中最有用的通用技术，因为它需要非常少的内存空间，而且在大多数计算机上，当很好地实现时，它的平均运行时间少于其它算法的平均运行时间。但是在最坏的情况下，它运行得非常之慢，所以当可能是非随机的数据时，应当小心选择分划的元素。如习题 5.2.2-55 中所建议的，选择 3 个元素的中间值，使最坏的情况极不可能发生，同时也对平均运行时间有所改进。

8. **直接选择，**算法 5.2.3S 当可用特殊硬件来高速寻道一个表的最小元素时，是一个特别合适的简单方法。

9. **堆排序，**算法 5.2.3H 需要极小的内存，而且保证十分快地运行；它的平均时间和极大时间都约为快速排序平均运行时间的两倍。

10. **表合并，**算法 5.2.4L　此算法是一个表排序，它和堆排序相像，保证甚至在最坏的情况下也是相当快的，而且对于相等的键码它是稳定的。

11. **基数排序，**使用算法 5.2.5R　它是一种表排序，特别适合于这样一些键码的排序，它们或者比较短，或者是按一个非通常的字典顺序排列。也可以使用分布计数方法(见上面的 1)以代替链接；这样一个过程需要 $2N$ 个记录空间，加上一个计数器表，但是它的内部循环的简单形式，使它特别适用于超高速的"吞嚼数据"的有先行控制的计算机。**警告：**基数排序不应对小 N 使用！

12. **合并插入，**见 5.3.1 小节　在一个"直接行编码"的例程中，特别适合于非常小的 N。例如，在需要对大量 5 个或 6 个记录一组排序的应用中，它将是适当的方法。

13. **混合的方法**　把上述的一种或多种技术组合在一起，也是可能的。例如，合

并插入可用于快速排序中出现的短的子文件排序。

14.最后,一个出现在习题 5.2.1-3 答案中的未命名方法,似乎要求最短的排序程序。但它的平均运行时间同 N^3 成正比,这就使它成为本书中最慢的排序例程!

表 1 综述了当为 MIX 编程时,这些方法的速度和空间特征。重要的是要认识到,这个表中的数字仅仅是相对的排序时间的粗略表示;它们仅应用于一台计算机上,而且关于输入数据所做的假定对所有的程序来说并不完全一致。其它作者给出了与此表相似的其它一些比较表,而且没有两个人给出相同的结论来!另一方面,这些估计时间至少给出当对一个字的记录排序时,对每种算法可能预期的大体速度,因为 MIX 是一台相当典型的计算机。

表 1 中的"空间"列,给出了每种算法所使用的辅助存储数量的某些信息,它以记录长度为单位,其中"ϵ"表示记录中链接字段部分所需的长度。例如,$N(1+\epsilon)$意味着这个方法要求 N 个记录加上 N 个链接字段的空间。

表 1 中出现的渐近平均时间和极大时间,仅仅给出了当 N 很大时占支配地位的前导项,且假定为随机输入;c 表示一个未确定的常数。这些公式可能常会引起误解,所以特地对两个特殊的输入数据序列,列出了程序实际的总运行时间。$N=16$ 的情况指的是 16 个键码,它出现在 5.2 节的许多示例中;$N=1000$ 的情况指的是由

$$K_{1001}=0;\ K_{n-1}=(3141592621K_n+2113148651)\bmod 10^{10}$$

所定义的序列 $K_1,K,\cdots,K_{1000}$。一个相当"高质量"的 MIX 程序用来表示表中的每种算法,通常还加上了习题中所建议的改进。这些程序运行时的字节大小为 100。

外部排序技术不同于内部排序,因为它们必须使用比较原始的数据结构,并且着重强调把它们的输入/输出时间极小化。5.4.6 小节综述了对于磁带合并已经发现的有趣方法,5.4.9 小节讨论了磁盘和磁鼓的作用。

当然,排序并不是我们的全部目的。在研究所有这些排序技术的同时,我们学习了大量的有关如何处理数据结构、如何处理外存,以及如何分析算法等问题;而且,也许我们甚至已经掌握了一点儿如何来发现新的算法的本领。

早期的发展 对于今天排序技术起源的探索,把我们带回到 19 世纪,因为那时发明了第一批用于排序的机器。美国每十年进行一次全国人口普查,而到 1880 年左右,处理庞大的人口普查数据的问题就变得十分尖锐;事实上,独身者(相对于结了婚的)的总数总不能在当年就造出表,即使所需要的信息都已经收集好了也是如此。Herman Hollerith,人口统计局的一名 20 岁的职员,发明了一台巧妙的电动制表机,以满足更好地进行统计收集的需要,他的大约 100 台机器成功地用于制造 1890 年的人口名册。

图 94 所示为 Hollerith 最初的用电池驱动的装置;我们的主要兴趣在于右边的"排序盒",它已被打开以示出 26 个内部小室的一半。操作员插入一张 $6\frac{5}{8}$In × $3\frac{1}{4}$ In 的穿孔卡片到"夹具"上,然后放下手柄;这使得在卡片中穿有孔的地方,上面板上

表 1　利用 MIX 计算机对内部排序方法进行的比较

方　法	参　见	稳定否?	MIX 程序的长　度	空　间	运行时间 平　均	运行时间 极　大	运行时间 $N=16$	运行时间 $N=1000$	注
比较计数	习题 5.2-5	是	22	$N(1+\epsilon)$	$4N^2+10N$	$5.5N^2$	1065	3992432	c
分布计数	习题 5.2-9	是	26	$2N+1000\epsilon$	$22N+10010$	$22N$	10362	32010	a
直接插入	习题 5.2.1-33	是	10	$N+1$	$1.5N^2+9.5N$	$3N^2$	412	1491928	
shell 排序	程序 5.2.1D	否	21	$N+\epsilon\lg N$	$3.9N^{7/6}+10N\lg N+166N$	$cN^{4/3}$	567	128758	d,h
表插入	习题 5.2.1-33	是	19	$N(1+\epsilon)$	$1.25N^2+13.25N$	$2.5N^2$	433	1248615	b,c
多表插入	程序 5.2.1M	否	18	$N+\epsilon(N+100)$	$0.0175N^2+18N$	$3.5N^2$	645	35246	b,c,f,i
合并交换	习题 5.2.2-12	否	35	N	$2.875N(\lg N)^2$	$4N(\lg N)^2$	939	284366	
快速排序	程序 5.2.2Q	否	63	$N+2\epsilon\lg N$	$11.67N\ln N-1.74N$	$\geqslant 2N^2$	470	81486	
3 个取中快速排序	习题 5.2.2-55	否	100	$N+2\epsilon\lg N$	$10.63N\ln N+2.12N$	$\geqslant N^2$	487	74574	e
基数交换	程序 5.2.2R	否	45	$N+68\epsilon$	$14.43N\ln N+23.9N$	272N	1135	137614	g,i,j
直接选择	程序 5.2.3S	否	15	N	$2.5N^2+3N\ln N$	$3.25N^2$	853	2525287	j
堆　排　序	程序 5.2.3H	否	30	N	$23.08N\ln N+0.01N$	$24.5N\ln N$	1068	159714	h,j
表　合　并	程序 5.2.4L	是	44	$N(1+\epsilon)$	$14.43N\ln N+4.92N$	$14.4N\ln N$	761	104716	b,c,j
基数表排序	程序 5.2.5R	是	36	$N+\epsilon(N+200)$	$32N+4838$	$32N$	4250	36838	b,c

a：仅 3 个数字的键码

b：仅 6 个数字(即 3 字节)键码

c：输出不重新排列;通过链接或计数器隐式地指明最后的序列

d：如同在 5.2.1-(11)中那样选择增量,习题 5.2.1-29 中出现稍微更好些的序列

e：$M=9$,利用 SRB;对于用 DIV 的版本,平均运行时间加上 $1.60N$

f：$M=100$(字节大小)

g：$M=34$,因为 $2^{34}>10^{10}>2^{33}$

h：平均时间以经验估计为基础,因为这一理论不完备

i：平均时间以一致分布的键码的假定为基础

j：正文中和伴随这一程序的习题提出的进一步改进,将减少运行时间

用弹簧驱动的针同在下面板上的水银容器相接触。相应的完整的线路将引起控制台面板上的相关转磁盘前进一个单位;然后,26 个排序盒盖之一推开。这时,操作员重新打开夹具,把卡片放到打开的小室,并关上盖。据说,一个人在 6 个半小时的一个工作日中通过这个机器能运行 19 071 张卡片,大约每分钟 49 张卡片$\left(\text{一个熟练操作员的工作速度大约仅为这个速度的}\frac{1}{3}\text{左右}\right)$!

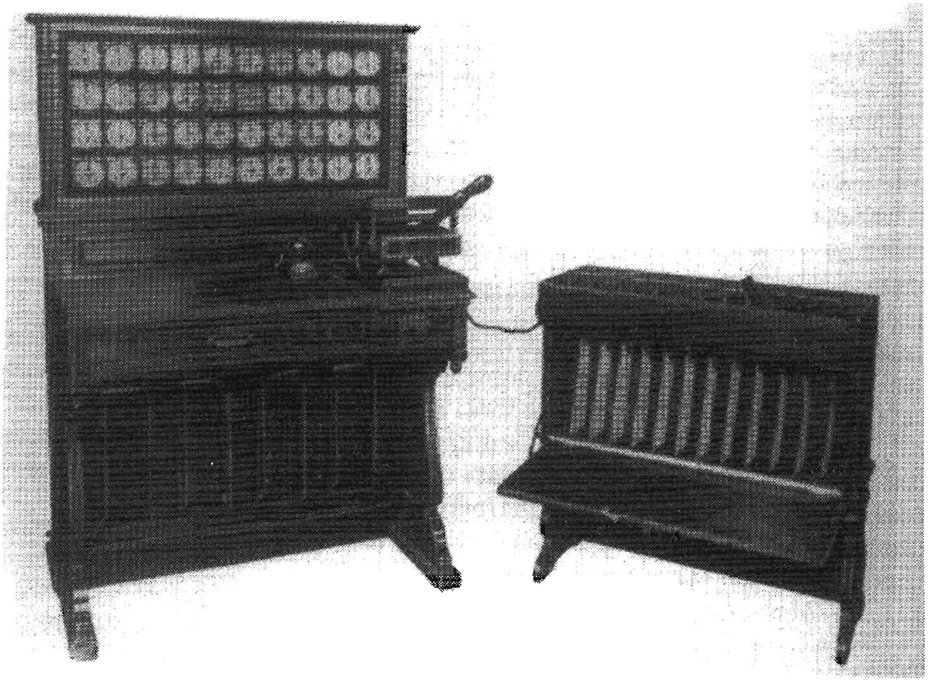


图 94 Hollerith 最初的制表机和排序机
(经 IBM 档案室同意采用此照片)

随着人口的持续增长,原来的制表排序机已经不足以处理 1900 年的人口,所以 Hollerith 又发明了另一台机器以解决数据处理的危机。他的新设备(1901 年和 1904 年的专利)有一个自动的进卡(片)器,事实上它很像现代的卡片排序机。关于 Hollerith 早期机器的故事,Leon E. Truesdell 曾在 *The Development of Punch Card Tabulation* (Washington: U. S. Bureau of the Census, 1965)中予以有趣而详尽地描述。另外也可参见当时的报导,如:*Columbia College School of Mines Quarterly* **10** (1889),238~255;*J. Franklin Inst.* **129**(1890),300~306;*The Electrical Engineer* **12** (November 11, 1891), 521 ~ 530;*J. Amer. Statistical Assn.* **2**(1891), 330 ~ 341, **4** (1895),365;*J. Royal Statistical Soc.* **55**(1892), 326 ~ 327;*Allgemeines statistisches Archiv* **2**(1892),78~126;*J. Soc. Statistique de Paris* **33**(1892),87~96;*U. S. Patents*

395781 (1889),*685608* (1901),*777209* (1904)。Hollerith 和人口统计局另一名职员 James Powers 接着又创办了互相竞争的公司，它们最终分别成为 IBM 和 Remington Rand 公司的一部分。

Hollerith 的排序机当然是现在数字计算机中基数排序方法的基础。他的专利提到，两列数值项的排序是对每列独立进行的，但他没有说是否应当首先考虑个位列或者十位列。

由 John K. Gore 在 1894 年获得的编号为 518240 的专利，描述了用于对于卡片排序的另一台早期的机器，它提议由十位的列开始。使用个位列的不明显的技巧大概首先是由某个不知名的机器操作员发现的，然后传给其他人(参见 5.2.5 小节)；它出现在最早期流行的 IBM 排序机手册(1936 年)中。这个自右至左技术的第一个已知的出处是在 Robert Feindler 所写的一本书*Das Hollerith-Lochkarten-Verfahren* (Berlin:Reimar Hobbing,1929),126～130 中。大约在同一时间，L. J. Comrie 在一篇论文*Transactions of the Office Machinery Users' Association* (London:1929—1930), 25～37 中也曾提到。很偶然地，Comrie 成了作出重要发现的第一人，这一发现就是，制表机最初尽管是为统计和结算而设计的，但它们可以有效地用于科学计算中。他的文章特别有趣，因为它给出了 1930 年在英国可买到的制表设备的详细描述。那时的排序机每分钟处理 300～400 张卡片，而且可以以每月 9￡的租金出租。

合并的思想可回溯到另一部卡片滚动机器：**整理机**，这是一项晚得多的发明(1938 年)。通过它的两个进卡站，可以在仅仅一次扫描中就把两叠排好序的卡片合并成一叠；完成此项工作的技术，在第一本 IBM 整理机手册(1939 年 4 月)中就清楚地说明了[参见 James W. Bryce,*U. S. Patent 2189024* (1940)]。

随后计算机走到了前台，而且终于把排序也包括在其发展过程当中。事实上有证据表明，排序例程曾经是为可存储程序的计算机编写的第一个程序。EDVAC 的设计者们对于排序特别感兴趣，因为它集中体现了计算机潜在的非数值应用；他们认识到，一组令人满意的指令代码，不仅应有能力表达用于解差分方程的程序，它还必须有足够的灵活性来处理算法中的组合“判定”问题。约翰·冯·诺依曼因此于 1945 年编制了用于内部合并排序的程序，为的是检验他建议的 EDVAC 计算机的某些指令代码的适用性；有效的专用排序机的存在，提供了一个自然的标准，通过这种标准就可以评价他所提议的计算机组织的优点。本书作者在*Computing Surveys* **2** (1970),247～260 一文中已经描述了这一有趣的发展细节。关于冯·诺依曼对最初的排序程序“润饰”后的形式，可参见冯·诺依曼的*Collected Works* **5**(New York: Macmillan,1963),196～214。

在德国，K. Zuse 于 1945 年独立编写了用于直接插入排序的一个程序，作为他的 Plankalkül 语言中线性列表操作的最简例子之一(这一开创性的工作推迟了近 30 年才发表；见*Berichte der Gesellschaft für Mathematik und Datenverarbeitung* **63** (Bonn:1972),Part 4,84～85。

由于为早期计算机设计的内存容量是很有限的，所以很自然地，既要考虑内部排序也要考虑外部排序。由穆尔电子工程学校的 J. P. Eckert 和 J. W. Mauchly 所撰

写的报告“Progress Report on the EDVAC”(1945 年 9 月 30 日)指出,装有磁线或磁带设备的一台计算机可以模拟卡片机的操作,达到更快的排序速度。这个报告描述了平衡的两路基数排序,以及平衡的两路合并(称为“整理”),同时使用 4 个磁线或磁带机,“每秒至少读或写 5000 个脉冲”。

1946 年在穆尔学校举行的有关计算的专题讨论会上,John Mauchly 作了“排序和整理”的演讲,他的演讲稿是第一个公开发表的关于计算机排序的讨论[*Theory and Techniques for the Design of Electronic Digital Computers*, G. W. Patterson 主编,**3** (1946),22.1～22.20]。Mauchly 以一段有趣的话来开始他的介绍:“要求一台机器把计算和排序的能力结合在一起,似乎很像要求一个设备既能用作开瓶器,又能用作自来水笔。”然后他说,有能力进行复杂的数学运算的机器必须也有能力对数据排序和分类,而且他还说排序甚至在数值计算中也会有用。他描述了直接插入和二叉插入,说明前一种方法平均使用大约 $N^2/4$ 个比较,而后一种方法则不会需要多于 $N\lg N$ 个比较。然而,二叉插入需要一个颇为复杂的数据结构,他接着说明两路合并仅仅使用对表的顺序存取就达到了同样低的比较次数。他的演讲稿的后半部分专门讨论了部分扫描的基数排序方法,即模拟 4 条磁带上的数字卡片排序,同时每位数字使用少于 4 次的扫描(参见 5.4.7 小节)。

不久以后,Eckert 和 Mauchly 创立了一个公司,这个公司生产了一些最早的电子计算机,BINAC(供军事应用)以及 UNIVAC(供商业应用)。美国人口统计局再次在其发展中发挥了作用,接受了第一台 UNIVAC。这时,还完全不了解计算机在经济上的作用;用计算机排序可以比卡片机快,但它们要昂贵得多。因此,由 Frances E. Holberton 领导的 UNIVAC 程序员们,花费相当大的精力设计高速的外部排序例程,并且他们早期的程序也影响了硬件的设计。按照他们的估计,在 UNIVAC 上一亿个 10 个字的记录可以在 9000h(即 375 天)之内完成排序。

UNIVAC Ⅰ正式交货于 1951 年 7 月,它的内存为有 1000 个字长 12 字符(72 位)的字。它被设计成以每秒钟 500 个字的速度,在磁带上读写 60 个字长的块;可以向前或向后读,而且可以同时进行读/写/计算。1948 年,Holberton 夫人想出了一个有趣的方法,使用 6 个输入缓冲区,以读、写和计算的完全重叠来进行两路合并:设每个输入文件都有一个“当前的缓冲区”和两个“辅助的缓冲区”,有可能以这样一种方式来进行合并,即每当输出一个块时,两个当前的输入缓冲区包含着总共恰恰相当一个块的未处理的记录。因此,在形成每个输出块时,恰有一个输入缓冲区变空。故我们可以安排成,在所有时刻,当要读入一个辅助缓冲区时,4 个辅助缓冲区中的另外 3 个都是满的。这个方法比算法 5.4.6F 的预报方法要稍微快些,因为它不需要在开始下一个输入之前,检查已输入的结果[参见“Collation Methods for the UNIVAC System”(Eckert-Mauchly Computer Corp.,1950),2 Volumes]。

这项工作的顶峰即排序生成程序,它是自动程序设计领域中研制的第一个重要的软件例程。用户指明记录的大小、每个记录的部分字段中最多 5 个键码的位置以及标记文件结束的哨兵键码,而排序生成程序将为一卷文件产生受版权保护的排序程序。此程序的第一遍扫描是使用比较计数(算法 5.2C)的 60 字块的一个内部排

序;然后进行一些平衡的两路合并扫描,向后读,以及避免如上所述的磁带互锁[参见“Master Generating Routine for 2-way Sorting”(Eckert-Mauchly Division of Remington Rand,1952)。这个报告的第一个草稿的标题为“Master Prefabrication Routine for 2-way Collation”。另外也可参见 F.E.Holberton 的*Symposium on Automatic Programming* (Office of Naval Research,1954),34～39]。

到 1952 年左右,内部排序的许多方法已在程序设计领域广为流传,但理论上的研究却相对很少。Daniel Goldenberg[“Time analyses of various methods of sorting data”,Digital Computer Laboratory memo M-1680(Mass. Inst. of Tach.,17 October 1952)]用 Whirlwind 计算机编写了 5 个不同方法的程序,并对每个程序都就最好的情况和最坏的情况进行了分析。当对 100 个具有 8 位键码的 15 位字进行排序时,他发现最快的方法是使用一个 256 字的表,把每个记录存入对应于它的键码的惟一位置,然后压缩这份表。但这项技术有一个明显的缺点,因为每当后继的记录有相同的键码时,它将冲掉一个记录。他所分析的其它 4 种方法排列如下:直接两路合并优于基数 2 排序优于直接选择优于冒泡排序。

Goldenberg 的结果,由 Harold H.Seward 在他 1954 年的硕士论文[“Information sorting in the application of electronic digital computers to business operations”,Digital Computer Lab. report R-232(Mass. Inst. of Tech.,24 May 1954;60 页)]中作了推广。Seward 引进了分布计数和替代选择的思想;他证明在一个随机排列中第一个路段的平均长度为$e-1$;而且他分析了在磁带上以及在各种类型的海量存储设备上的内部排序以及外部排序。

由 Howard B.Demuth 于 1956 撰写的博士论文〔“Electronic Data Sorting”(Stanford University,october 1956 年 10 月),92 页;*IEEE Trans.* **C-34** (1985),296～310〕,可以说一篇非常值得关注的论文,因为这篇论文有助于奠定计算复杂性理论的基础。它利用循环的、线性的以及随机存取的存储器,考虑了排序问题的 3 个抽象模型,并对每个模型提出了最优的或接近于最优的方法(参见习题5.3.4-68)。尽管 Demuth 的论文并没有立即产生实际的结果,但它却建立了如何把理论同实践相联系的重要思想。

这样一来,排序的历史已经同计算中的许多“第一”紧密地联系在一起:第一个数据处理机器,第一个可存储程序,第一个软件,第一个缓存方法,第一个进行算法分析和计算复杂性分析。

迄今所述的有关计算机的文献,实际上都未出现在“公开的著作”中;事实上,计算的大多数早期的历史,都出现在比较难以得到的报告中,因为在那时仅有比较少的人跟计算机打交道。有关排序文献的第一次付印是在 1955 年～1956 年,用的是 3 篇重要的综述性文章。

第一篇综述性文章由 J.C.Hosken 撰写[*Proc. Eastern Joint Computer Conference* **8** (1955),39～55]。他以敏锐的观察开始:“为降低每个输出单位的价格,人们通常都在增加他们的操作规模。但在这样的条件下,排序的单位费用,不是降低了,而是增加了。”Hosken 综述了在计算机上进行排序的方法,以及所有可利用的现在已在

销售的专用设备。他的 54 项参考文献目录大多数是以厂家的手册为基础的。

E.H.Friend 的内容丰富的论文 Sorting on Electronic Computer Systems[*JACM* **3**(1956),134～168],是排序发展中的一个重要里程碑。尽管自 1956 年以来已经发展了许多技术,但这篇论文在许多方面直到今天仍然值得注意。Friend 对于相当多的内部和外部排序算法给出了细致的描述,而且他对于缓存技术和磁带机的特征给以特殊的关注。他引进了某些新的方法(例如,树的选择,两头蛇排序以及预报),而且展示了较早方法的某些数学性质。

大约在同一个时间里出现的第三篇关于排序的综述性文章,是由 D.W.Davies [*Proc. Inst. Elect. Engineers* **103B**, Supplement 1(1956),87～93]撰写的。在随后数年中,下列作者也发表了许多值得关注的综述:D.A.Bell[*Comp. J.* **1**(1958),71～77];A.S.Douglas[*Comp. J.* **2**(1959),1～9];D.D.McCracken,H.Weiss 以及 T.Lee(李再华)[*Programming Business Computers* (New York:Wiley,1959),第 15 章,298～332];I.Flores[*JACM* **8**(1961),41～80];K.E.Iverson[*A Programming Language* (New York:Wiley,1962),第 6 章,176～245];C.C.Gotlieb[*CACM* **6**(1963),194～201]:T.N.Hibbard[*CACM* **6**(1963),206～213];M.A.Goetz [*Digital Computer User's Handbook*, M.Klerer 和 G.A.Korn 主编(New York:McGraw-Hill,1967),1.10 章,1.292～1.320]。1962 年 11 月 ACM 主持召开了一次关于排序的研讨会;在该会上宣读的大多数论文都发表在*CACM* 1963 年 5 月的刊物上,并且它们乃是当时技术发展水平的很好代表。C.C.Gotlieb 关于现代排序生成程序的综述,T.N.Hibbard 关于极小存储内部排序的综述,以及 G.U.Hubbard 关于磁盘文件排序的早期剖析,都是这个文集中特别值得关注的文章。

在整个这段时期还发现了一些新的排序方法:地址计算(1965 年),合并插入(1959 年),基数交换(1959 年),级联合并(1959 年),Shell 排序(1959 年),多阶段合并(1960 年),树插入(1960 年),振荡排序(1962 年),Hoare 的快速排序(1962 年),William 的堆排序(1964 年),Batcher 的合并交换(1964 年)。在本章特定小节中,在描述方法的同时已经追述了每个算法的历史。20 世纪 60 年代后期,相应的理论得到了深入的发展。

当最初写本章时,作者所考察的有关排序所有论文的一份完整参考文献,是在 R.L.Rivest 的帮助下编辑的,出现于*Computing Reviews* **13**(1972),283～289 中。

未来发展 自 1970 年以来,已有数十个排序算法被发明出来,尽管几乎所有的这些算法都是早期算法的变种。在习题 5.2.2-30 的答案中讨论的多键码快速排序,是这些新方法中的一个出色例子。

另外一个趋势,迄今主要还是理论上的,是研究可自适应的排序方案。即在这样一个意义下,按照不同原则,当输入已经很趋于有序时,能够保证它们运行得更快。请参见 H.Mannila,*IEEE Transactions* **C-34**(1985),318～325;V.Estivill-Castro 和 D.Wood,*Computing Surveys* **24**(1992),441～476;C.Levcopoulos 和 O.Petersson,*Journal of Algorithms* **14**(1993),395～413;A.Moffat,G.Eddy 和 O.Petersson,*Soft-*

ware *Practice & Experience* **26**(1996),781～797。

当开销的准则发生变化时,计算机硬件的改变促进了关于排序算法有效性的许多有趣研究。例如,习题 5.4.2-20 中关于虚拟内存的讨论。A. LaMarca 和 R. E. Ladner 分析了硬件的高速缓存对于内部排序的影响(*SODA* **8**(1997),370～379)。他们的结论之一是,在现代机器上算法 5.2.2Q 的步骤 Q9 是一个不好的想法(尽管在像 MIX 这样的传统计算机上它工作得很好):应代之以直接插入排序来完成快速排序,此时最好早些对短的子文件进行排序,因为这时它们的关键字仍然在高速缓存中。

当前对于大量数据进行排序的技术其发展水平又如何呢?自 1985 年以来,一个普遍的标准是对 100 万个含 100 个字符的记录进行排序,而这些记录有完全随机的 10 字符的关键字。输入和输出应当驻留在磁盘上,目标是极小化总的消逝的时间,包括用于启动程序的时间。R. C. Agarwal [*SIGMOD Record* **25**,2(June 1996),240～246] 使用一台桌面 RISC 计算机,即 IBM RS/6000 型 39H,对被分片到 8 个磁盘机的文件实现基数排序,它在 5.1s 内完成这一任务。输入/输出是主要的瓶颈;确实,处理器只需要 0.6s 来控制实际的排序!当有多个处理器可以利用时,甚至可以更快。用 32 台 Ultra SPARC I 工作站组成一个网络,每个工作站有两个内部的磁盘,使用一个称为 NOW-Sort 的混合方法,能在 2.41s 内对 100 万个记录进行排序 [A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, D. E. Culler, J. M. Hellerstein 和 D. A. Patterson,*SIGMOD Record* **26**,2(June1997),243～254]。

这样一些进展意味着百万个记录的标准变成主要是启动和关闭时间的一个测试,所以需要更大的数据集合以给出更有意义的结果。例如,太拉字节排序——10^{10} 个含有 100 个字符的记录——现在的世界纪录是 2.5h。它是由一个具有 32 个处理器的 Silicon Graphics Origin 2000 系统在 1997 年 9 月实现的,其中每个处理器有 8GB 的内存,以及 559 个 4GB 的磁盘。这个记录是由一个市场上销售的称为 Nsort™的排序程序创造的,它是由 C. Nyberg, C. Koester 以及J. Gray使用尚未公开的方法编写的。

也许有一天太拉字节也会被认为是太小的标准。一个标准要想永远有效,目前的候选者就是分钟排序:即在 60s 之内可以对多少个 100 字符的记录进行排序?在本书付印时,对于这一任务当前记录的保持者是 NOW-Sort:1997 年 3 月 30 日,95 个工作站只需 59.21s 就把 90.25 百万个记录排好了序。但是现今的方法并没有真正提高速度的根本极限。

总之,今天有效排序的问题仍然和它曾经有过的一样,是一个迷人的问题。

习 题

1.[*05*] 试通过叙述定理 5.4.6A 的一个推广,来总结这一章的内容。

2.[*20*] 基于表 1 中的信息,对于 6 位数的键码,什么是 MIX 计算机上最好的表排序方法?

3.[*37*](在极小存储中的稳定排序) 我们说一个排序算法需要极小的存储空间,如

果除了存储 N 个记录所需的空间外，它的变量仅使用 $O((\log_2 N)^2)$ 位存储空间。这个算法在下列意义下必须是通用的，即必须对所有的 N 都有效，而不仅仅对一个具体的 N 值有效，其中假定当实际调用这个算法来进行排序时，已经有充分数量的随机存取内存可资利用。

我们已经研究过的许多排序方法，都违背了这个极小存储的要求；特别是，使用 N 个链接字段是禁止的，快速排序（算法 5.2.2Q）满足极小存储要求，但它最坏情况的运行时间竟同 N^2 成正比。堆排序（算法 5.2.3H）是我们所研究过的惟一使用极小存储 $O(N\log_2 N)$ 的算法，尽管利用习题 5.2.4-18 的思想，还可描述另一个这样的算法。

我们前面所研究的、以一种稳定的方针对键码排序的、最快的一般算法是表合并排序（算法 5.2.4L），但它不使用极小存储。事实上，我们已经见到的稳定的极小存储排序算法都是 $\Omega(N^2)$ 的方法（直接插入，冒泡排序以及直接选择的一种变种）。

试设计一个稳定的极小存储排序算法，它在最坏的情况下仅需 $O(N(\log N)^2)$ 个时间单位［提示：有可能在 $O(N\log N)$ 个时间单位内进行稳定的极小存储合并］。

▶4.［*28*］如果一个排序算法完全通过比较键码来做决定，而且它从来不做这样一个比较，即此比较的结果可以通过以前比较的结果预测出来，则这样一个排序算法称为是吝啬的。问表 1 中列出的方法中哪些是吝啬的？

5.［*46*］排序具有许多相等键码的非随机数据，比排序完全随机的数据要困难得多。试设计一个排序标准使得：(i)它现在是有效的，而且从现在起大概 100 年后仍然是有效的；(ii)它不包括完全随机的键码；(iii)它不使用随时间而改变的数据集。

I shall have accomplished my purpose if I have sorted and put in logical order the gist of the great volume of material which has been generated about sorting over the past few years.

将过去若干年得到的关于排序的大量材料的要旨分好类排好序之日，也即是我的目标实现之时。

—— J.C.HOSKEN(1955)

第6章 查　　找

Let's look at the record
让我们来查看这个记录。
——AL Smith(1928)

这一章可以给以更大的标题"信息的存储和检索";另一方面,它也可以简单地称做"查表"。我们关心的是在一台计算机的存储中收集信息的过程,以及随后尽可能快地进行对该信息的检索。有时,我们面对比我们实际需要的还要多的数据,因此最明智的办法是把这些数据的大部分忘掉或销毁;但在其它情况下,重要的是,要以能进行快速检索的方式保留和组织给定的数据。

这一章大部分篇幅都致力于研究一个非常简单的查找问题:怎样根据某种确定的标记来找出存储好的数据。例如,在一个数值应用中,给定了 x 和一张 f 值表后要求 $f(x)$;在一个非数值应用中,可能要找一个给定的俄文单字的英语翻译。

一般说来,我们将假设,N 个记录的一个集合已经存储好,问题是要找出所要的记录。如同排序那样,假定每个记录都包括称为它的*键码*的一个特殊字段,取这个名字也许是因为许多人每天都要花很多时间来找他们的钥匙。一般我们都要求 N 个键码互不相同,以便每一个键码惟一地标识它的记录。所有记录的集合称为一份*表*或一个*文件*,"表"在这里通常用来表示一个小文件,而"文件"通常用来表示一个大的表。一个大的文件或一组文件经常称做一个*数据库*。

查找算法中有一个变元 K,问题是要寻找以 K 为其键码的那一个记录。在查找完成之后,有两种可能性:或者这个查找是*成功的*,已找到含有 K 的惟一记录,或者是*不成功的*,已确定 K 无处可找。在不成功的查找之后,有时希望送入包含 K 的一个新记录到这份表中,做这项工作的方法称为查找和插入算法。名叫"联想存储"的某些硬件设备,以模拟人脑功能的方式自动地解决查找的问题;而我们将研究在一部通常的通用计算机上来进行查找的一些技术。

尽管查找的目标是寻找存储在同 K 相关联的记录中的信息,但这一章中的算法一般都忽略除键码本身以外的一切。实际上,一旦我们定出 K 的地址,便可找出相关联的数据;例如,如果 K 出现在 TABLE + i 的位置中,则相关联的数据(或指向该数据的一个指针)可在单元 TABLE + $i+1$ 中,或在单元 DATA + i 中等。因此,找到 K 之后,不妨把应该做什么等细节都忽略掉。

查找是许多程序中最消耗时间的一部分,因而用一个好的查找方法来替代一个坏的方法,常常会使运行速度大大提高。事实上,我们通常能把数据或数据结构安

排好,可完全免去查找,确保我们知道上哪儿去找我们需要的信息。链接存储是实现这一点的常用的方法。例如,使用双重链接表就不必查找一个给定项的前驱或后继。如果允许我们自由地选择键码,则是查找的另一个方法,因为我们也可以令诸键码为数$\{1,2,\cdots,N\}$;于是包含 K 的记录就可简单地放置在单元 TABLE + K 中。在 2.2.3 小节讨论的拓扑排序算法中,这两项技术都曾被用来消除查找。然而,如果拓扑排序算法中的对象是符号名而不是数,仍然需要查找。在实践中用于查找的有效算法变得十分重要的。

查找方法可用若干种方式分类。我们可以把它们分为内部查找和外部查找,如同把第 5 章的排序算法分为内部排序和外部排序一样。或者可以把查找方法分为静态查找和动态查找,其中“静态”指的是表的内容实际上不变(可以使查找时间减少到最小而不考虑建立表所需要的时间),而“动态”指的是这个表经常要受到插入或许删除的影响。第三种可能的方案是,按照查找方法是以键码之间的比较为基础还是以键码的数字性质为基础来分类,就像用比较进行排序和用分布进行排序之间的区别那样。最后,我们还可以根据使用实际的键码还是使用变换了的键码来对查找方法分类。

这一章的组织,实质上是后两种分类方式的综合。6.1 节考虑查找的“硬找”的顺序查找的方法,6.2 节讨论了一种改进的方法,它以键码之间的比较为基础,根据字母或数字的顺序作出判断;6.3 节讨论数字查找;6.4 节讨论一类重要的方法,称做散列技术,它是以实际键码的算术变换为基础的。每一节都在静态和动态两种情况下,既讨论内部查找也讨论外部查找;并且每节都指出各种算法的相对优点和缺点。

查找和排序通常是彼此紧密相关的。例如,考虑以下的问题:*给定两组数* $A=\{a_1,a_2,\cdots,a_m\}$ 和 $B=\{b_1,b_2,\cdots,b_n\}$,*确定是否* $A\subseteq B$。这本身提示了三种解,即:

1. 顺序地把每个 a_i 同诸 b_j 作比较直到找到一个相同的为止。

2. 把诸 a 和诸 b 排序,然后作通过这两个文件的一个序列,校验适当的条件。

3. 把诸 b 记入一个表中,然后查找每一个 a_i。

上述每一种解法,在 m 和 n 值的不同范围内都是有吸引力的。对于某个常数 c_1,解 1 将花费大约 c_1mn 个时间单位,而对于某个(更大的)常数 c_2,解 2 将花费大约 $c_2(m\lg m+n\lg n)$个时间单位,对于某个(仍然很大的)常数 c_3 和 c_4,使用一个适当的散列方法,解 3 将花费大约 c_3m+c_4n 个时间单位。由此得出,对于很小的 m 和 n,解 1 是好的,但当 m 和 n 增大时,解 2 很快将变成更好。最后,解 3 变成可取的,直到 n 超过了内存大小为止,然后,解 2 往往又变得优越,至到 n 变得非常大为止。于是我们有这样一种情况:排序有时是查找的一个好替换,而查找有时又是排序的一个好替换。

更复杂的查找问题通常都可以归结成这里所考虑的较简单的情况。例如,假设键码可以是稍有拼错的字;而可能要在忽略这种错误的情况下仍能找到正确的记录。如果我们作一个文件的两个副本,一个副本中键码以通常的字母顺序出现,另

一个则按字母从右到左排好(好像单词是倒过来拼那样),则拼错的查找变元大概将同这两个文件之一中的一个项一致到它长度的一半或一半以上。因此 6.2 节和 6.3 节的查找方法可被用来寻找大概想要的键码。

一个有关的问题已受到了相当大的注意,它联系到飞机票预约系统及与人名有关的其它应用,在这些应用中,经常有可能由于拙劣的手写或声音传输而拼错名字。目标是要把变元转换成某个代码,这个代码有助于把同一名字的所有变形都转换成同一形式。下述的"探测"(Soundex)方法,最初是由 Margaret K. Odell 和 Robert C. Russell 提出的。[参见*U. S. Patents* 1261167(1918),1435663(1922)。]它常常用来对姓氏编码:

1. 保留名字的头一个字母,并且省略在其余位置中出现的所有 a,e,h,i,o,u,w,y。

2. 把下列数字赋给头一个字母之后的剩余的字母

b,f,p,v→1	l→4
c,g,j,k,q,s,x,z→2	m,n→5
d,t→3	r→6

3. 如果具有相同代码的两个或多个字母在原来的名字(在步骤 1 前)中相邻,或者除开干扰诸 h 和诸 W 之外相邻,则省略除头一个以外的所有字母。

4. 通过增加尾部零(如果少于三个数字),或者省略最右边的数字(如果有三个以上的数字),转换成"字母,数字,数字,数字"的形式。

例如,Euler,Gauss,Hilbert,Knuth,Lloyd,Lukasiewicz 以及 Wachs 这些名字的代码分别是 E460,G200,H416,K530,L300,L222,W200。当然,这个系统会把有些不同的名字或类似的名字变成同一形式;Ellery,Ghosh,Heibronn,Kant,Liddy,Lissajous 以及 Waugh 就得到同样的七个代码。另一方面,一些有关的名字,如 Rogers 和 Rodgers,Sinclair 和 St. Clair,Tchebysheff 和 Chebyshev,则仍保持互异。但总的说来,Soundex 代码大大地增加了在伪装的代码中找出名字的机会。[更多的介绍,参考 C. P. Bourne 和 D. F. Ford, *JACM* 8 (1961), 538~552; Leon Davidson, *CACM* 5(1962),169~171;*Federal Population Censuses* 1790~1890(Washington D. C.:National Archives,1971),90。]

当使用像 Soundex 这样的方案时,不必放弃所有键码都不同这样的假定;我们可以造出具有等价代码的所有记录的表组,把每一个表当成一个单位来处理。

大型数据库势必使检索过程更为复杂,因为人们通常要把每个记录的许多不同的字段都当作可能的键码,当只知道部分键码信息时,也有能力来定出项目的位置。例如,给定关于戏剧演员的一个大文件,舞台监督可能希望找出 25 岁到 35 岁之间,具有跳舞天才和法国口音的所有还没有角色的女演员;给定垒球统计的一个大文件,一个体育专栏作家可能希望确定 1964 年在进行晚场比赛的第七个回合,对抗左手投手时芝加哥的 White Sox 跑出的总分数。给定关于任何事情的一个大型数据文件后,人们会任意地提出复杂的问题。确实,我们可以把整个图书馆看成一个数据库,而查找者可能要寻道已发表的有关情报检索的每份资料。在下面的 6.5 节

中,将介绍用于此类辅助键码(多重属性)检索问题的技术。

在对查找进行详细研究之前,有必要回顾一下历史。在计算机出现以前,就编辑出版了对数表、三角函数表等许多书,使得数学计算可为查表所代替。最后,这些表被誊写到穿孔卡片上,并用于同整理机、排序机以及拷贝穿孔机相联系的科学问题中。但一旦出现了存储程序的计算机,很显然:每次重新计算 $\log x$ 或 $\cos x$ 比在一张表中查出答案要更合算些。

尽管在计算机出现伊始,排序问题就受到了相当大的关注,但是对于查找的算法,工作却做得比较少。由于内存小和只有磁带这样的顺序介质能用来存储大型文件,因此查找要么极为容易,要么几乎不可能。

但是自 20 世纪 50 年代起,越来越大的随机存取存储器的发展,最终导致了这样一个认识,即查找就其本身的性质而言是一个有趣的问题。在抱怨了好几年早期计算机的空间局限性之后,程序员们突然面对着非常大的内存容量,以致不知道如何有效使用它。

关于查找问题最初的一些综述文章如下:A. I. Dumey, *Computers & Automation* **5**,12(1956 年 12 月),6～9;W. W. Peterson, *IBM J. Research & Development* **1**(1957),130～146;A. D. Booth, *Information and Control* **1**(1958),159～164;A. S. Douglas, *Comp. J.* **2**(1959),1～9。后来,下面文章对查找问题进行了更广泛的讨论:Kenneth E. Iverson, *A Programming Language* (New York: Wiley, 1962),133～158,和 Werner Buchholz, *IBM Systems J.* **2**(1963),86～111。

如我们将要看到的,在 20 世纪 60 年代初期,引进了许多有趣的新的基于树结构的查找方法;而关于查找的研究现在仍然活跃地开展着。

6.1 顺序查找

"从起点开始往下查,直到找到了正确的键码为止;然后停止"。这个顺序过程是进行查找的明显方法,这个方法也构成了我们讨论查找问题的一个有用的起点,因为许多更错综复杂的算法都是以它为基础的。我们将看到,顺序查找除了简单之外,还包含了某些非常有趣的思想。

这个算法可以更精确地阐述如下。

算法 S(顺序查找) 给定其键码分别为 $K_1, K_2, \cdots, K_N$ 的记录 $R_1, R_2, \cdots, R_N$ 的一个表,这个算法查找一个给定的变元 K,假定 $N \geqslant 1$。

S1.[初始化] 置 $i \leftarrow 1$。

S2.[比较] 如果 $K = K_i$,则此算法成功地结束。

S3.[前进] i 增加 1。

S4.[文件结尾?] 如果 $i \leqslant N$,则返回 S2。否则此算法以失败告终。 ▎

注意,这个算法可以以两种不同的方式结束,即*成功*(已定出所需键码的位置),或*不成功*(已证实给定的变元不在该表中)。这一章中大多数其余的算法都同样如

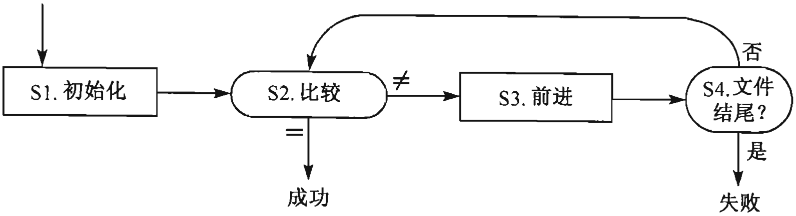


图 1　顺序查找或“逐字”查找

此。

一个 MIX 程序可立即写出来。

程序 S（顺序查找）　假定 K_i 出现于单元 KEY + i 中，记录 R_i 的剩余部分出现在单元 INFO + i 中，下列程序使用 $rA \equiv K$，$rI1 \equiv i - N$。

```
START     LDA     K            1        S1.初始化
          ENT1    1 - N        1        i←1
2H        CMPA    KEY + N,1    C        S2.比较
          JE      SUCCESS      C        如果 K = Ki,转出
          INC1    1            C - S    S3.前进
          J1NP    2B           C - S    S4.文件结尾?
FAILURE   EQU     *            1 - S    如果不在表中,则转出
```

现在，在单元 SUCCESS 处指令“LDA INFO + N,1”将把所需信息带到 rA 中。▌

对这个程序的分析很直截了当；它表明算法 S 的运行时间依赖于两个因素

$$C = \text{键码比较的次数；}$$
$$S = 1\text{，如果成功；}0\text{，如果不成功。} \tag{1}$$

程序 S 花费 $5C - 2S + 3$ 个时间单位。如果这个查找成功地找到 $K = K_i$，则有 $C = i$，$S = 1$；因此总的时间为 $(5i + 1)u$。另一方面，如果查找不成功，则有 $C = N$，$S = 0$，总的时间为 $(5N + 3)u$。如果每个输入键码都以相同的概率出现，则在一次成功的查找中，C 的平均值将是

$$\frac{1 + 2 + \cdots + N}{N} = \frac{N + 1}{2} \tag{2}$$

当然标准差肯定相当大，约为 $0.289N$（见习题 1）。

上述算法所有程序员都熟悉，但是，很少有人知道，它并不总是进行顺序查找的正确方式！只要作一些直接修改就会使这个方法更快，除非记录表很短。

算法 Q（快速顺序查找）　这个算法和算法 S 相同，惟一的区别是它假定在这个文件的结尾存在一个“虚拟”记录 R_{N+1}。

Q1.［初始化］　置 $i \leftarrow 1$，并置 $K_{N+1} \leftarrow K$。

Q2.［比较］　如果 $K = K_i$，转到 Q4。

Q3.［前进］　i 增加 1，并返回 Q2。

Q4.[文件结尾?] 如果 $i \leqslant N$，则算法成功地结束；否则以失败告终($i = N + 1$)。▌

程序 Q(快速顺序查找) $rA \equiv K$, $rI1 = i - N$。

```
START    LDA    K            1              Q1.初始化
         STA    KEY+N+1      1              K_{N+1}←K
         ENT1   -N           1              i←0
         INC1   1            C+1-S          Q3.前进
         CMPA   KEY+N,1      C+1-S          Q2.比较
         JNE    *-2          C+1-S          如果 K_i≠K 转到 Q3
         J1NP   SUCCESS      1              Q4.文件结尾?
FAILURE  EQU    *            1-S            若不在表中则转出 ▌
```

用量 C 和 S 来分析程序 S，运行时间已减少到$(4C - 4S + 10)u$；在一个成功的检索中每当 $C \geqslant 6$ 时，或者在一个不成功的检索中，每当 $N \geqslant 8$ 时它都是一个改进。

从算法 S 到算法 Q 的过渡利用了一个重要的"加速"原理：当程序中的一个内循环要测试两个或多个条件时，则应该力图将测试减少为只有一个条件。

另一项技术将使程序 Q 还要快。

程序 Q′(更快的顺序查找) $rA \equiv K$, $rI1 \equiv i - N$。

```
START    LDA    K            1              Q1.初始化
         STA    KEY+N+1      1              K_{N+1}←K
         ENT1   -1-N         1              i←-1
         INC1   2            ⌊(C-S+2)/2⌋    Q3.前进(两步)
         CMPA   KEY+N,1      ⌊(C-S+2)/2⌋    Q2.比较
         JE     4F           ⌊(C-S+2)/2⌋    若 K=K_i 则转到 Q4
         CMPA   KEY+N+1,1    ⌊(C-S+1)/2⌋    Q2.比较(第二个)
         JNE    3B           ⌊(C-S+1)/2⌋    如果 K≠K_{i+1} 则转
                                            到 Q3
         INC1   1            (C-S)mod 2     i 前进
4H       J1NP   SUCCESS      1              Q4.文件结尾?
FAILURE  EQU    *            1-S            如不在表中则转出 ▌
```

内循环已被重复；这避免了大约一半的"$i \leftarrow i + 1$"指令，所以它使运行时间减少为

$$3.5C - 3.5S + 10 + \frac{(C - S)\bmod 2}{2}$$

个单位。当正被查找的是很大的表时，我们已经节省了程序 S 运行时间的 30%；许多现存的程度都能以这种方式改进。同样的方法适用于高级语言的编程。[例如，

参考 D.E.Knuth(*Computing Surveys*)6(1974),266～269]

如果我们知道键码是递增次序的,则应稍微改变一下算法。

算法 T（在有序表中顺序查找）　给定一个其键码为递增次序 $K_1 < K_2 < \cdots < K_N$ 的记录 $R_1, R_2, \cdots, R_N$ 的表,这个算法查找一个给定的变元 K。为了方便,和快捷,本算法假定有一个键码值为 $K_{N+1} = \infty > K$ 的虚拟记录 R_{N+1}。

T1.[初始化]　置 $i \leftarrow 1$。

T2.[比较]　如果 $K \leqslant K_i$,转到 T4。

T3.[前进]　i 加 1,并返回 T2。

T4.[相等?]　如果 $K = K_i$,则此算法成功地终止,否则,它以失败告终。　▌

如果所有的输入键码都是同等可能的,则当查找成功时,这个算法所花的时间实质上和算法 Q 的平均运行时间相同。但当查找不成功时它比算法 Q 快大约两倍,因为该算法可以更快地确定一个记录不存在。

上述每一种算法都使用下标表示表的项。借助于下标来描述这些方法是很方便的,但是同样的查找过程也可用于一个使用链式表示的表中,这是因为数据是被顺序地遍历的(见习题 2、3 和 4)。

存取频率　至今我们一直假定每个变元都以同样频率出现。这并不总是一个现实的假定;一般情况下,键码 K_i 将以概率 p_i 出现,其中,$p_1 + p_2 + \cdots + p_N = 1$。为进行一个成功的查找所需要的时间实际上同比较的次数 C 成正比,C 的平均值为

$$\overline{C}_N = p_1 + 2p_2 + \cdots + Np_N \tag{3}$$

若我们能以任意想要的次序把记录放置到表中,则当

$$p_1 \geqslant p_2 \geqslant \cdots \geqslant p_N \tag{4}$$

时,即当最经常使用的记录出现在接近开始处时量 $\overline{C}_N$ 最小。

现在让我们考虑若干概率分布,以了解当诸记录以(4)中所确定的"最优"方式排列时,可能节省多少时间? 若 $p_1 = p_2 = \cdots = p_N = 1/N$,则公式(3)简化为 $\overline{C}_N = (N+1)/2$;我们已经在等式(2)中导出这个值。另一方面,假设

$$p_1 = \frac{1}{2}, p_2 = \frac{1}{4}, \cdots, p_{N-1} = \frac{1}{2^{N-1}}, p_N = \frac{1}{2^{N-1}} \tag{5}$$

则由习题 7,$\overline{C}_N = 2 - 2^{1-N}$;若诸记录在表中以正确的次序出现,则对于这个分布,比较的平均数小于 2。

另一个可供考察的概率分布是

$$p_1 = Nc, \quad p_2 = (N-1)c, \quad \cdots, \quad p_N = c$$

其中

$$c = 2/N(N+1) \tag{6}$$

这个"楔形的"分布不像(5)那样厉害地偏离一致分布。在这种情况下,我们发现

$$\bar{C}_N = c\sum_{k=1}^{N} k(N+1-k) = \frac{N+2}{3} \tag{7}$$

即此种最优排列比起诸记录以随机顺序出现时节省了大约三分之一的查找时间。

当然,(5)和(6)中的概率分布是相当人为的,因而它们可能不是现实的非常好的近似。一个更为典型的分布是“Zipf(吉甫)定律”

$$p_1 = c/1, p_2 = c/2, \cdots, p_N = c/N, \text{其中 } c = 1/H_N \tag{8}$$

这个分布是由 G. K. Zipf 得到的。他发现在用自然语言写的文章中第 n 个最常用的单字似乎近似地以与 $1/n$ 成正比的概率出现[参考 *The Psycho-Biology of Language*(Boston, Mass.: Houghton Mifflin, 1935); *Human Behavior and the Principle of Least Effort* (Reading, Mass: Addison - Wesley, 1949)]。他发现当大城市按递减人口数排列时,人口统计表中也有同样的现象。如果吉甫定律适用于一个表中键码的频率,则我们立即有

$$\bar{C}_N = N/H_N \tag{9}$$

查找这样一个文件大约比查找按随机次序排列记录的同一文件快$\frac{1}{2}\ln N$ 倍。

[参考 A. D. Booth, L. Brandwood 以及 J. P. Cleave, *Mechanical Resolution of Linguistic Problems* (New York: Academic Press, 1958), 79。]

对于现实分布的另一个近似,是在商业应用中已被普遍地观察到的“80-20”估计规则[参考 W. P. Heising, *IBM System J.* **2**(1963), 114～115]。这个规则指出,80%的事务涉及文件最常用的20%;而对这20%适用同样的规则,则使得64%的事务涉及最活跃的4%,等等,换言之

$$\frac{p_1 + p_2 + \cdots + p_{.20n}}{p_1 + p_2 + p_3 + \cdots + p_n} \approx .80 \quad \text{对所有 } n \tag{10}$$

当 n 是5的倍数时,恰好满足这个规则的一个分布是

$$p_1 = c,\ p_2 = (2^\theta - 1)c,\ p_3 = (3^\theta - 2^\theta)c,\ \cdots,\ p_N = (N^\theta - (N-1)^\theta)c \tag{11}$$

其中

$$c = 1/N^\theta, \quad \theta = \frac{\log .80}{\log .20} \approx 0.1386 \tag{12}$$

因为在这种情况下,对于所有的 n, $p_1 + p_2 + \cdots + p_n = cn^\theta$。要分析(11)中的概率不太容易;然而,由于我们有 $n^\theta - (n-1)^\theta = \theta n^{\theta-1}(1 + O(1/n))$,所以有一个更简单的近似满足80-20规则的分布,即

$$p_1 = c/1^{1-\theta},\ p_2 = c/2^{1-\theta},\ \cdots,\ p_N = c/N^{1-\theta},\ \text{其中 } c = 1/H_N^{(1-\theta)} \tag{13}$$

如前,这里 $\theta = \log 80/\log 20$,且 $H_N^{(s)}$ 是阶为 s 的第 N 个调和数,即 $1^{-s} + 2^{-s} + \cdots + N^{-s}$。注意,这个概率分布非常类似于 Zipf 定律(8);当 θ 从 1 变成 0 时,这个概率分布从一个一致分布变成一个 Zipf 分布。应用(3)到(13)得出

$$\bar{C}_N = H_N^{(-\theta)}/H_N^{(1-\theta)} = \frac{\theta N}{\theta + 1} + O(N^{1-\theta}) \approx 0.122N \tag{14}$$

作为关于 80-20 定律的平均比较数(见习题 8)。

由 E. S. Schwartz 进行的单字频率的研究[参考 *JACM* **10** (1963),422 页上有趣的图],提示磁带有稍微负的 θ 值的分布(13)给出比 Zipf 规则(8)对数据更好的适合,在这种情况下,当 $N\to\infty$ 时均值

$$\bar{C}_N = H_N^{(-\theta)}/H_N^{(1-\theta)} = \frac{N^{1+\theta}}{(1+\theta)\zeta(1-\theta)} + O(N^{1+2\theta}) \tag{15}$$

比(9)要小得多。

类似(11)和(13)的分布,首先是由 Vilfredo Pateto 在同个人收入和对财富的不一致性相关联中研究的[参考*Cours d'Économie Politique* **2** (Lausanne: Rouge, 1897), 304～312]。如果 p_k 与第 k 个最富有的人的财富成比例,当 $x = p_k/p_N$ 时一个人的财富超过或等于最穷的人的 x 倍的概率是 k/N。因此,当 $p_k = ck^{\theta-1}$ 和 $x = (k/N)^{\theta-1}$ 时所述的概率是 $x^{-1/(1-\theta)}$。现在把这叫做磁带有参数 $1/(1-\theta)$ 的 Pareto 分布。

奇怪的是,Pareto 不理解他自己的分布;他相信,接近于 0 的一个 θ 值要比接近于 1 的 θ 值对应于更平等的社会!他的错误是由 Corrado Gini 改正的[参考*Atti della Ⅲ Riunione della Società Italiana per il Progresso delle Scienze* (1910),在*Memorie di Metodologia Statistica1* (Rome: 1955),3～120 重印]。他是第一个陈述和阐释像 80-20 定律(10)的比的重要性的人。人们仍然倾向于误解这样的分布;他们通常谈到"75 − 25 定律"或"90 − 10 定律",就好像仅当 $a+b=100$ 时,$a-b$ 定律才有意义,而(12)表明,80 + 20 的和是十分无所谓的。

类似于(11)和(13)的另一个离散分布是由 G. Udny Yule 在假定各种进化模型时,研究作为时间函数的生物品种的增加时引进的[参考*Philos. Trans.* B213 (1924),21～87]。当 $\theta<2$ 时,Yule 的分布适用:

$$p_1 = c,\ p_2 = \frac{c}{2-\theta},\ p_3 = \frac{2c}{(3-\theta)(2-\theta)},\cdots,p_N = \frac{(N-1)!c}{(N-\theta)\cdots(2-\theta)} = \frac{c}{\binom{N-\theta}{N-1}}$$

$$c = \frac{\theta}{1-\theta}\cdot\frac{\binom{N-\theta}{N}}{1-\binom{N-\theta}{N}} \tag{16}$$

当 $\theta=0$ 或 $\theta=1$ 时极限值 $c=1/H_N$ 或 $c=1/N$。

一个"自组织"的文件　上述对于概率的计算看起来挺不错,但是在大多数情况下并不知道概率是多少。我们可以在每个记录中建立一个计数器来记住它被存取的次数,在这些计数的基础上重新分配记录;上边导出的公式告诉我们,这个过程通常能节省不少运行时间。但是或许我们不想提供那么多的存储空间来作计数字段,因为通过利用在这一章稍后将要说明的非顺序查找技术,我们可以更好地利用内

存。

有一种简单的方案,可用来把各记录按一种相当好的次序存放而无需使用附加的计数字段。尽管并不知道这个方案的来源,但它已经使用多年了:每当一个记录被成功地定址时,便把它移到表的开头。

这个“自组织”技术的思想背景是,当需要查找常用的项目时,应在相当接近于表的开头处找到它们。若假定 N 个键码分别以概率 $\{p_1, p_2, \cdots, p_N\}$ 出现,而每次查找又完全独立于先前的查找,则可以证明,在这样一个自组织的文件中为寻道一个项目所需要的比较的平均数趋于极限值

$$\widetilde{C}_N = 1 + 2\sum_{1\leqslant i<j\leqslant N}\frac{p_i p_j}{p_i + p_j} = \frac{1}{2} + \sum_{i,j}\frac{p_i p_j}{p_i + p_j} \tag{17}$$

(见习题 11)。例如,若当 $1\leqslant i\leqslant N$ 时有 $p_i = 1/N$,则自组织表总是以完全随机的次序出现,且这个公式简化为上面所熟悉的公式 $(N+1)/2$。一般,比较的平均数(17)总是小于最优值(3)的两倍,因为 $\widetilde{C}_N \leqslant 1 + 2\sum_{j=1}^{N}(j-1)p_j = 2\overline{C}_N - 1$。事实上,$\widetilde{C}_N$ 总是小于 $\pi/2$ 乘最优值 $\overline{C}_N$ [参考 Chung, Hajela, 和 Seymour, *J. Comp Syst Sci*, **36** (1988), 148~157];一般,这个比是最好的常数,因为当 p_j 和 $1/j^2$ 成正比时,它是接近的。

现在让我们看看当键码的概率服从 Zipf 定律(8)时,自组织过程的效果如何。由等式 1.2.7-(8)和 1.2.7-(3),我们有

$$\begin{aligned}\widetilde{C}_N &= \frac{1}{2} + \sum_{1\leqslant i,j\leqslant N}\frac{(c/i)(c/j)}{c/i + c/j} = \frac{1}{2} + c\sum_{1\leqslant i,j\leqslant N}\frac{1}{i+j} = \\ &\frac{1}{2} + c\sum_{i=1}^{N}(H_{N+i} - H_i) = \frac{1}{2} + c\sum_{i=1}^{2N}H_i - 2c\sum_{i=1}^{N}H_i = \\ &\frac{1}{2} + c((2N+1)H_{2N} - 2N - 2(N+1)H_N + 2N) = \\ &\frac{1}{2} + c(N\ln 4 - \ln N + O(1)) \approx 2N/\lg N\end{aligned} \tag{18}$$

当 N 相当大时,这比 $\frac{1}{2}N$ 要好得多,而且它仅仅为最优排列中所得到比较次数的大约 $\ln 4 \approx 1.386$ 倍;参见(9)。

有关实际编译程序符号表的计算经验指出,自组织方法的效果甚至比我们的公式所预料的更好,因为逐次的查找并不是独立的(小批的键码趋向于成串出现)。

John McCabe[*Operations Research* **13** (1965), 609~618]首先分析了这个自组织方案,他建立了(17)。McCabe 还介绍了另一个有趣的方案,其中每个成功地找到的、已不在表开头的关键字,只是简单地与前面的键码互换,而不是总被移到前端。他猜想,假定是独立查找,这一方法的极限平均查找时间决不超过(17)。事实上,后来 Ronald L. Rivest 证明,除了当 $N\leqslant 2$ 时或当所有非零概率都相等之外(这是当然

的)，此换位方法严格地使用比移至前头方法要少的比较[*CACM* **19** (1976)，63－67]。然而，对渐近极限的收敛要比移向前端的磁带启发式的查找要慢得多，所以除非这一过程被拖延，否则移向前端的方法更好，[J. R. Bitner *SICOMP* **8** (1979)，82～110]。况且，J. L. Bentley，C. C. McGeoch，D. D. Sleator 和 R. E. Tarjan 已经证明，给定对于数据的任何访问序列——即使算法知道未来，移向前端的方法决不会做比任何关于线性表的算法所作的内存访问总数超出四倍的内存访问；频率计数和交换位置方法就没有这个属性[*CACM* **28** (1985) 202～208，404～411]。有关由 R. Bachrach 和 R. El _ Yaniv 所作的关于自组织表多于 40 种的带启发式查找的有趣的经验研究，参见*SODA* **8** (1997) 53～62。

对于不等长记录的磁带查找　现在让我们把问题提升到一个新的难度：假设正在查找的表存储在磁带上，并且个别记录有可变的长度。例如，在一个旧式的操作系统中，“系统库磁带”就是这样一个文件；标准的系统程序，诸如编译程序、汇编程序、装入程序、报告生成程序等，都是这条磁带上的“记录”，而大多数用户作业一开始就要从头至尾查找此磁带，直到读入适当的程序为止。这种体制使我们以前对算法 S 的分析已不适用。因为步骤 S3 在每次执行时所花费的时间是变化的。因此，比较次数并不是惟一重要的准则。

设 L_i 是记录 R_i 的长度，p_i 是记录被找到的概率，则这个查找方法的平均运行时间近似地同

$$p_1L_1 + p_2(L_1 + L_2) + \cdots + p_N(L_1 + L_2 + L_3 + \cdots + L_N) \tag{19}$$

成正比。当 $L_1 = L_2 = \cdots = L_N = 1$ 时，此式简化为(3)，这是已经研究过了的情况。

把最经常需要的记录放在磁带的开头好像是合乎逻辑的；但有时这却是个坏主意！例如，假定磁带仅包含两个程序 A 和 B；对 A 的需要是对 B 的需要的两倍，而 A 的长度却是 B 的四倍。于是，$N = 2$，$p_A = \frac{2}{3}$，$L_A = 4$，$p_B = \frac{1}{3}$，$L_B = 1$。如果首先把 A 放在磁带上，按照上面所述“合逻辑的”原理，则平均运行时间是 $\frac{2}{3}\cdot 4 + \frac{1}{3}\cdot 5 = \frac{13}{3}$；但如果使用一个“不合逻辑”的想法，首先放置 B，则平均运行时间就被减少到 $\frac{1}{3}\cdot 1 + \frac{2}{3}\cdot 5 = \frac{11}{3}$。

在库磁带上程序的最优排列可以确定如下：

定理 S　*设 L_i 和 p_i 定义如上，则当且仅当*

$$p_1/L_1 \geqslant p_2/L_2 \geqslant \cdots \geqslant p_N/L_N \tag{20}$$

时表中记录的排列为最优。换言之，当且仅当(20)成立时，对于$\{1,2,\cdots,N\}$的所有排列 $a_1a_2\cdots a_N$，$p_{a_1}L_{a_1} + p_{a_2}(L_{a_1} + L_{a_2}) + \cdots + p_{a_N}(L_{a_1} + \cdots + L_{a_N})$的极小值等于(19)。

证明　假设 R_i 和 R_{i+1}在磁带上互换；则代价(19)就从

$$\cdots + p_i(L_1 + \cdots + L_{i-1} + L_i) + p_{i+1}(L_1 + \cdots + L_{i+1}) + \cdots$$

变成 $$\cdots + p_{i+1}(L_1 + \cdots + L_{i-1} + L_{i+1}) + p_i(L_1 + \cdots + L_{i+1}) + \cdots$$

纯变化为 $p_iL_{i+1} - p_{i+1}L_i$。因此，如果 $p_i/L_i < p_{i+1}/L_{i+1}$，则这样一个交换将改进平均运行时间，因而给定的排列就不是最优的。由此得出，(20)在任何最优的排列中成立。

反之，假定(20)成立；我们需要证明排列是最优的。刚才给出的论证表明，这个排列是"局部地最优的"，其意义是交换相邻元素不会导致改进；但可以设想有一个长的复杂的交换序列，它导出一个更好的"全局优化"。我们将考虑两个证明，一个证明使用计算机科学，一个使用一个数学技巧。

第一个证明 假定(20)成立。我们知道，通过相邻记录的一系列交换，记录的任何排列可以被排成次序 $R_1R_2\cdots R_N$。每次交换都对某个 $i<j$ 以$\cdots R_iR_j\cdots$代替$\cdots R_jR_i\cdots$，所以它使查找时间减少了非负量 $p_iL_j - p_jL_i$。因此次序 $R_1R_2\cdots R_N$ 必定有极小的查找时间。

第二个证明 把每个概率 p_i 代之以

$$p_i(\epsilon) = p_i + \epsilon^i - (\epsilon^1 + \epsilon^2 + \cdots + \epsilon^N)/N \tag{21}$$

其中 ϵ 是极其小的正数。当 ϵ 充分小时，决不可能有 $x_1p_1(\epsilon) + \cdots + x_Np_N(\epsilon) = y_1p_1(\epsilon) + \cdots + y_Np_N(\epsilon)$，除非 $x_1 = y_1\cdots, x_N = y_N$；特别是，等式在(20)中将不成立。现在考虑记录的 $N!$个排列；至少其中有一个是最优的，且知道它满足(20)；但是由于没有等式，故仅有一个排列满足(20)。因此只要 ϵ 充分小，(20)就惟一地表征了概率为 $p_i(\epsilon)$的表中诸记录的最优排列。由连续性，当 ϵ 被置为 0 时，同一排列必然也是最优的(在与组合最优化有关的问题中，这种"解结"类型的证明通常是有用的)。▌

定理 S 是由 W.E.Smith 在"*Naval Research Logistics Quarterly* **3**(1956)，59～66中给出的。下面的习题包含了有关最优文件安排进一步的结果。

习 题

1.[*M20*] 当所有的查找键码都有相同的查找概率时，在对一个有 N 个记录的表进行成功的顺序查找中，所作比较次数的标准差是多少？

2.[*15*] 利用链接存储记号代替下标记号，重新叙述算法 S 的诸步骤(如果 P 指向表中的一个记录，则假定 KEY(P)是键码，INFO(P)是相关的信息，并假定 LINK(P)是指向下一个记录的指针，FIRST 指向头一个记录，最后的记录指向 Λ)。

3.[*16*] 写出习题 2 中算法的一个 MIX 程序。如果用(1)中的量 C 和 S 来表达，你的程序运行时间是多少？

▶4.[*17*] 算法 Q 的思想是否可不用下标记号而用链接存储记号实现(参见习题 2)？

5.[*20*] 当 C 很大时，程序 Q′当然比程序 Q 快得多。但是否有任何小的 C 和 S 值，使程序 Q'实际上比程序 Q 花的时间多？

▶6.[*20*] 对程序 Q′添加三条以上的指令，使它的运行时间减少到大约 $(3.33C+\text{常数})u$。

7.[*M20*] 试利用“二进的”概率分布(5)计算比较的平均数(3)。

8.[*HM22*] 当 $x\neq 1$ 时，试求 $n\to\infty$ 时 $H_n^{(x)}$ 的一个渐近级数。

▶9.[*M28*] 正文指出，当 $0<\theta<1$ 时，由(11)、(13)和(16)给出的概率分布大致是相等的，而且利用(13)的平均比较数是 $\frac{\theta}{\theta+1}N+O(N^{1-\theta})$。

a)当使用(11)的概率时，平均比较数是否也等于

$$\frac{\theta}{\theta+1}N+O(N^{1-\theta})$$

b)(16)等于多少？

c)当 $\theta<0$ 时，(11)和(16)如何同(13)进行比较？

10.[*M20*] (4)确定了在一个顺序表中诸记录最好的排列；什么是最坏的排列？证明在最坏的排列中的平均比较数与最好的排列中的平均比较数的简单关系。

11.本题的目的是分析通过移向前端的磁带启发式查找的一个自组织文件的极限特性。首先我们需要定义一些记号：设 $f_m(x_1,x_2,\cdots,x_m)$ 是所有不同的有序积 $x_{i_1}x_{i_2}\cdots x_{i_k}$ 的无穷和，其中 $1\leqslant i_1,\cdots,i_k\leqslant m$ 且每个 $x_1,x_2,\cdots,x_m$ 都出现在每一项中。例如

$$f_2(x,y)=\sum_{j,k\geqslant 0}(x^{1+j}y(x+y)^k+y^{1+j}x(x+y)^k)=$$
$$\frac{xy}{1-x-y}\left(\frac{1}{1-x}+\frac{1}{1-y}\right)$$

给定 n 个变量 $\{x_1,x_2,\cdots,x_n\}$ 的集合 X，令

$$P_{nm}=\sum_{1\leqslant j_1<\cdots<j_m\leqslant n}f_m(x_{j_1},\cdots,x_{j_m});\quad Q_{nm}=\sum_{1\leqslant j_1<\cdots<j_m\leqslant n}\frac{1}{1-x_{j_1}-\cdots-x_{j_m}}$$

例如，$P_{32}=f_2(x_1,x_2)+f_2(x_1,x_3)+f_2(x_2,x_3)$，$Q_{32}=1/(1-x_1-x_2)+1/(1-x_1-x_3)+1/(1-x_2-x_3)$。我们约定置 $P_{n0}=Q_{n0}=1$。

a)假定正文中的自组织文件已以概率 p_i 响应对项目 R_i 的需求，在这个系统运行了很长一段时间之后，试证 R_i 将以极限概率 $p_iP_{(N-1)(m-1)}$ 成为从前端起的第 m 个项目，其中

$$X=\{p_1,\cdots,p_{i-1},p_{i+1},\cdots,p_N\}$$

b)对 $m=1,2,\cdots$，把(a)的结果加起来，我们就得到恒等式

$$P_{nn}+P_{n(n-1)}+\cdots+p_{n0}=Q_{nn}$$

由此证明

$$P_{nm}+\binom{n-m+1}{1}p_{n(m-1)}+\cdots+\binom{n-m+m}{m}p_{n0}=Q_{nm}$$

$$Q_{nm}-\binom{n-m+1}{1}Q_{n(m-1)}+\cdots+(-1)^m\binom{n-m+m}{m}Q_{n0}=P_{nm}$$

c)计算 R_i 与表前端的极限平均距离 $d_i=\sum_{m\geqslant 1}mp_iP_{(N-1)(m-1)}$；然后求值

$$\widetilde{C}_N=\sum_{i=1}^{N}p_id_i$$

12.[*M23*] 当查找键码有二进概率分布(5)时，用(17)计算为查找自组织文件所需要的比较平均数。

13.[*M27*] 用(17)计算楔形概率分布(6)的 $\widetilde{C}_N$。

14.[*M21*] 给定两个实数序列 $\langle x_1,x_2,\cdots,x_n\rangle$ 和 $\langle y_1,y_2,\cdots,y_n\rangle$，什么样的下标排列 $a_1a_2\cdots$

a_n 将使 $\sum_i x_i y_{a_i}$ 成为极大？什么样的成为极小？

▶15.[*M22*] 前文指出，当仅寻到一个程序时，怎样在一条“系统库磁带”上最优地安排诸程序。但对于一条子程序库磁带，使用另一组假定更为适合，因为我们希望从这条磁带把一个用户程序中调用的诸子程序一起都装入用户程序中。

对于这种情况，我们假定需要子程序 j 的概率是 P_j，而不论是否需要其它子程序。例如全然不需要任何子程序的概率是 $(1-P_1)(1-P_2)\cdots(1-P_N)$；而查找恰在装入第 j 个子程序之后结束的概率是 $P_j(1-P_{j+1})\cdots(1-P_N)$。如果 L_j 是子程序 j 的长度，则平均的查找时间实质上将同

$$L_1P_1(1-P_2)\cdots(1-P_N)+(L_1+L_2)P_2(1-P_3)\cdots(1-P_N)+\cdots$$
$$+(L_1+L_2+\cdots L_N)P_N$$

成正比。在这些假定下，子程序在磁带上的最优排列是什么？

16.[*M22*] H. Riesel 我们通常需要测试：是否给定的 n 个条件同时全部为真(例如，我们可能要测试是否 $x>0$ 和 $y<z^2$ 两者都成立，而且首先应测试哪一个条件并不是一目了然的)。假设花费 T_j 个时间单位测试条件 j，而该条件为真的概率是 p_j，且同所有其它条件的结果无关。试问我们应在什么顺序下来进行测试？

17.[*M23*] (W. E. Smith) 假设你要做 n 个作业，第 j 个作业花费 T_j 个时间单位，且截止时间为 D_j。换言之，第 j 个作业应该在至多过了 D_j 个时间单位之后完成。试问为处理这些作业，什么样的调度 $a_1a_2\cdots a_n$ 将使极大的延迟

$$\max(T_{a_1}-D_{a_1},T_{a_1}+T_{a_2}-D_{a_2},\cdots,T_{a_1}+T_{a_2}+\cdots+T_{a_n}-D_{a_n})?$$

极小化？

18.[*M30*] (连接查找) 假定 N 个记录被放置在一个线性数组 $R_1\cdots R_N$ 中，记录 R_j 被查找的概率为 p_j。若每次查找都在刚才离开处开始，则这个查找过程称为“连接的”。如果诸连续的查找都是独立的，则所需的平均时间为 $\Sigma_{1\leqslant i,j\leqslant N}p_ip_jd(i,j)$，其中 $d(i,j)$ 表示从位置 i 处开始并在位置 j 处结束的一次查找所需的时间。若 $d(i,j)$ 为从圆柱面 i 扫视到圆柱面 j 所需的时间，则这个模型可应用到例如磁盘文件的寻道时间上。

本题的目的是每当 $d(i,j)$ 是 $|i-j|$ 的一个递增函数时，即每当对于 $d_1<d_2<\cdots<d_{N-1}$，我们有 $d(i,j)=d_{|i-j|}$ 时，(d_0 的值没有关系)，来表征连接查找诸记录的最优设置。

在这种情况下，证明在所有 $N!$ 个排列当中，各记录被最优排列的充分必要条件是 $p_1\leqslant p_N\leqslant p_2\leqslant p_{N-1}\leqslant\cdots\leqslant p_{\lfloor N/2\rfloor+1}$ 或者 $p_N\leqslant p_1\leqslant p_{N-1}\leqslant p_2\leqslant\cdots\leqslant p_{\lceil N/2\rceil}$。(这样，概率的一种“管风琴”排列是最好的，如图 2 所示)。提示：对于 $m\geqslant 0,k>0,N=2R+m+1$ 考虑任一排列，其中概率分别为 $q_1q_2\cdots q_ksr_k\cdots r_2r_1t_1t_2\cdots t_m$。证明若重新排列 $q_1'q_2'\cdots q_k'sr_k'\cdots r_2'r_1't_1\cdots t_m$，则效果更好，其中 $q_j'=\min(q_i,r_i)$，$r_i'=\max(q_i,r_i)$，除非对所有 i，$q_i'=q_i$ 和 $r_i'=r_i$，或对所有 i,j，$q_i'=r_i$，$r_i'=q_i$ 和 $t_j=0$。当 s 不出现且 $N=2k+m$ 时亦然。

19.[*M20*](继续习题 18) 当函数 $d(i,j)$ 有这样一个性质：即对于所有 $i\neq j$，$d(i,j)+d(j,i)=c$ 时，什么是连接查找的最优排列？[例如，当我们不知道查找的适当方向时，在没有回读能力的磁带上就出现这种情况。比如，对于 $i<j$，我们有，$d(i,j)=a+b(L_{i+1}+\cdots+L_j)$ 和 $d(j,i)=a+b(L_{j+1}+\cdots+L_N)+r+b(L_1+\cdots+L_i)$，其中，$r$ 为重绕时间。]

20.[*M28*] (继续习题 18) 对于 $d_1<d_2<\cdots$，当函数 $d(i,j)=\min(d_{|i-j|},d_{n-|i-j|})$ 时，什么是连接查找的最优排列？[例如：在两路链接的循环表中，或者在一个两路移位寄存器的存储设备上就出现这种情况。]

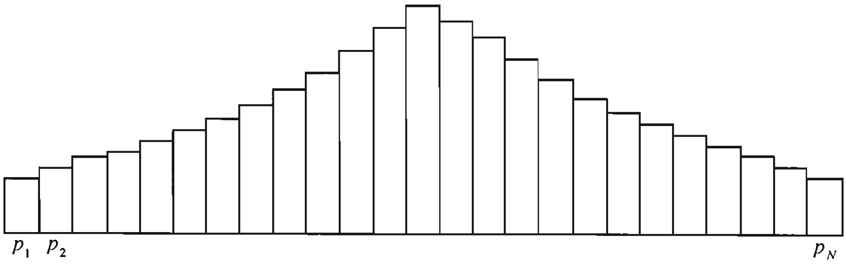


图 2 概率的“管风琴排列”使一个连接查找的平均寻找时间极小化

21.[M28] 考虑一个 n 维立方体,其顶点坐标为$(d_1,\cdots,d_n)$,且 $d_i=0$ 或 1;两个有一坐标不同的顶点称为相邻的。假设有 2^n 个数 $x_0\leqslant x_1\leqslant\cdots\leqslant x_{2^n-1}$的一个集合,以使$\sum_{i,j}|x_i-x_j|$成为极小这样一种方式,赋于 2^n 个顶点,其中的求和是对于所有的 i 和 j 进行的,而且要使 x_i 和 x_j 是赋给相邻顶点的两个数。试证明:若对所有 j,x_j 都被赋给一个这样的顶点,该顶点坐标的二进表示恰为 j,则达到上述极小值。

▶22.[20] 假设你要查找一个大型文件,不是查找相等的,而是找出最接近于一个给定键码的 1000 个记录。最接近的意义是:对于某个给定的距离函数 d,这 1000 个记录的$d(K_i,K)$值为最小。试问对于这样一个顺序查找,什么数据结构最为适合?

Attempt the end; and never stand to doubt;
Nothing's so hard, but search will find it out.
尝试宜穷尽,半途勿犹豫;天下无难事,探索将获解。
——ROBERT HERRICK *Seeke and finde* (1648)

6.2 通过键码比较进行查找

在这一节中,我们将讨论以键码的一个线性次序,例如,以字母顺序或数值顺序为基础的查找方法。在对给定的变元 K 和表中的一个键码 K_i 进行比较之后,分别取决于 $K<K_i$,$K=K_i$或$K>K_i$,这个查找以三种不同的方式继续。6.1 节的顺序查找方法实际上局限于两路判断($K=K_i$ 和$K\neq K_i$)。但如果我们摆脱顺序存取的限制,就可以有效地利用一个次序关系。

6.2.1 查找一个有序的表

如果某人递给你一本大电话簿,而且请你找一个人的名字,此人的电话号码是 795－6841,那你该做什么呢? 处理这个问题,没有比 6.1 节的顺序方法更好的方法了。(是的,你可以尝试拨这个号码并且同回话者谈;或者你可能知道如何找到以号码而不是以名字排序的一本特殊的电话簿。)问题在于通过当事者的名字而不是通过号码来找一个项目要容易得多,尽管在电话目录中包含了这两种情况下所需要的所有信息。当必须查找一个大型文件时,顺序扫描几乎是办不到的,而一个次序关系却大大地简化了这项工作。

使用已经讨论过的许多排序方法(第 5 章),把文件重新排为有序的,使其便于

查找,这不会太困难。当然,如果只需对这个表查找一次,则进行顺序查找比对这个文件进行完整的排序要快;但如果要在同一文件中进行重复查找,那么,更好的方法是把它按序排列。因此在这一节中,我们将集中研究这样一些方法,假定我们能够容易地存取在任一个给定位置的键码,则这些方法适于查找其键码满足

$$K_1 < K_2 < \cdots < K_N$$

的表。在这样的表中,比较 K 和 K_i 后,我们有

- $K < K_i$[不必考虑 $R_i, R_{i+1}, \cdots, R_N$],或
- $K = K_i$[查找完成],或
- $K > K_i$[不必考虑 $R_1, R_2, \cdots, R_i$]

除非 i 靠近该表末尾,这三种情况均已获实质性进展,这就是为什么次序能引出一个有效的算法的原因。

二分查找 最先想到的一个方法是从 K 同表的中间键码比较开始的;这个探查的结果指出下一次应该查找表的哪一半,并可再次使用相同的步骤,比较 K 和选中的一半的中间键码,等等。在至多进行了大约 $\lg N$ 次比较之后,或者找到这个键码,或者确认它不存在。这个步骤有时称为"对数查找"或"二等分法",通常称为二分查找,如图 3 所示。

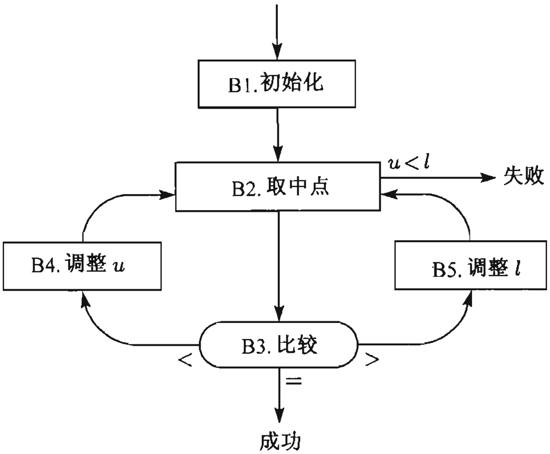


图 3 二分查找

尽管二分查找的思想比较直截了当,但其细节可能令人惊讶地复杂,而且许多好的程序员在他们头几次试用时竟把它弄错了。这个算法的最普遍的正确形式是利用两个指针 l 和 u 来指出当前查找的下限和上限如下:

算法 B(二分查找) 给定其键码在递增次序下 $K_1 < K_2 < \cdots < K_N$ 的记录 $R_1, R_2, \cdots, R_N$ 的一个表,本算法查找一个给定的变元 K。

B1.[初始化] 置 $l \leftarrow 1, u \leftarrow N$。

B2.［取中点］（这时我们知道如果 K 在此表中，则它满足 $K_l \leqslant K \leqslant K_u$。下面习题 1 中有关于这种情况的一个更精确的陈述。）如果 $u < l$，则这个算法以失败告终。否则，置 $i \leftarrow \lfloor (l+u)/2 \rfloor$，这是该表区域的近似中点。

B3.［比较］ 如果 $K < K_i$，转到 B4；如果 $K > K_i$，转到 B5；如果 $K = K_i$，则算法成功地结束。

B4.［调整 u］ 置 $u \leftarrow i-1$，并返回 B2。

B5.［调整 l］ 置 $l \leftarrow i+1$，并返回 B2。 ▌

图 4 说明了这个该二分查找算法的两种情况，第一个是查找变元 653，它在表中出现，而后查找 400，它不存在。方括号指出 l 和 u，下边划线的键码表示 K_i。在这两个例子中，进行了四次比较之后，查找结束。

a）查找 653

[061	087	154	170	275	426	503	$\underline{509}$	512	612	653	677	703	765	897	908]
061	087	154	170	275	426	503	509	[512	612	653	$\underline{677}$	703	765	897	908]
061	087	154	170	275	426	503	509	[512	$\underline{612}$	653]	677	703	765	897	908
061	087	154	170	275	426	503	509	512	612	[$\underline{653}$]	677	703	765	897	908

b）查找 400

[061	087	154	170	275	426	503	$\underline{509}$	512	612	653	677	703	765	897	908]
[061	087	154	$\underline{170}$	275	426	503]	509	512	612	653	677	703	765	897	908
061	087	154	170	[275	$\underline{426}$	503]	509	512	612	653	677	703	765	897	908
061	087	154	170	[$\underline{275}$]	426	503	509	512	612	653	677	703	765	897	908
061	087	154	170	275]	[426	503	509	512	612	653	677	703	765	897	908

图 4　二分查找的例子

程序 B（二分查找） 如同在 6.1 节的程序中那样，我们这里假定 K_i 是出现在单元 KEY + i 中的全字长键码。下列代码使用 $\mathrm{rI1} \equiv l$，$\mathrm{rI2} \equiv u$，$\mathrm{rI3} \equiv i$。

START	ENT1	1	1	*B1. 初始化* $l \leftarrow 1$
	ENT2	N	1	$u \leftarrow N$
	JMP	2F	1	转到 B2
5H	JE	SUCCESS	$C1$	如果 $K = K_i$ 则转移
	ENT1	1,3	$C1 - S$	*B5. 调整* l，$l \leftarrow i+1$
2H	ENTA	0,1	$C+1-S$	*B2. 取中点*
	INCA	0,2	$C+1-S$	rA←1 + u
	SRB	1	$C+1-S$	rA←⌊rA/2⌋（rX 也改变）
	STA	TEMP	$C+1-S$	
	CMP1	TEMP	$C+1-S$	
	JG	FAILURE	$C+1-S$	如果 $u < l$，则转移

```
12          LD3    TEMP        C     i←中点
13   3H     LDA    K           C     B3. 比较
14          CMPA   KEY,3       C
15          JGE    5B          C     如果 K≥K_i 则转移
16          ENT2   -1,3        C2    B4. 调整 u, u←i-1
17          JMP    2B          C2    转到 B2            ▌
```

这个步骤不像我们见过的其它算法那样同 MIX 十分“光滑地”融成一体，因为 MIX 不允许在变址寄存器中作许多算术运算。运行时间为 $(18C-10S+12)u$，其中 $C=C1+C2$ 是所作的比较数(即步骤 B3 被执行的次数)，以及 $S=[$结果是成功的$]$。在这个程序的 08 行处的“右移一个二进位”仅在 MIX 的二进制版本上是合法的；对于一般的字节大小，这条指令应该以“MUL = 1//2 + 1 =”代替，从而把运行时间增加到 $(26C-18S+20)u$。

树表示 为了真正理解在算法 B 中所发生的事情，最好把它想像成一株二分判定树，如图 5 所示，图中 $N=16$。

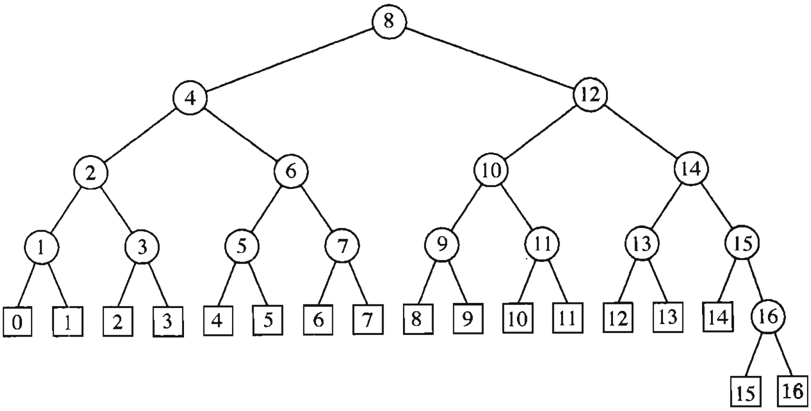


图 5 $N=16$ 时与二分查找对应的比较树

当 N 为 16 时，这个算法所作的头一个比较是 $K:K_8$；通过图中的根节点⑧来表示。然后，如果 $K<K_8$，则这个算法沿着左子树，比较 K 与 K_4；类似地，如果 $K>K_8$，则使用右子树。一个不成功的查找将通向编号为 $\boxed{0}$ 到 $\boxed{N}$ 的一个“外部”方形节点；例如，我们达到节点 $\boxed{6}$ 当且仅当 $K_6<K<K_7$。

在 N 个记录上进行二分查找的二叉树可以构造如下：如果 $N=0$，则这个树是 $\boxed{0}$，否则根节点是

$$\lceil N/2 \rceil$$

左子树是有$\lceil N/2\rceil-1$ 个节点对应的二叉树，右子树是有$\lceil N/2\rceil$个节点对应的二叉树，而且所有节点号都增加$\lceil N/2\rceil$。

类似地，任何借助于比较法查找长度为 N 的一个有序表的算法，都可以表示成一株 N 节点的二叉树，其中节点用 1～N 编号(除非算法作多余的比较)。反之，任何二叉树都对应一个查找有序表的有效方法；我们简单地以对称次序从左到右地对诸节点编号

$$\boxed{0}\quad ①\quad \boxed{1}\quad ②\quad \boxed{2}\quad \cdots\quad \boxed{N-1}\quad Ⓝ\quad \boxed{N} \tag{1}$$

如果在算法 B 中输入的查找变元是 K_{10}，则此算法进行 $K>K_8, K<K_{12}, K=K_{10}$的比较。这对应于图 5 中从根到⑩的通路。类似地，对其它键码来说，算法 B 的行为对应于从这株树的根导出的其它通路。因此，构造对应于算法 B 的二叉树的方法，使我们易于对 N 用归纳法证明下列结果。

定理 B　如果 $2^{k-1}\leqslant N<2^k$，*则利用算法 B 进行一个成功的查找需要作*(min 1, max k)*次比较。如果* $N=2^k-1$，*则一个不成功的查找需要 k 次比较；如果* $2^{k-1}\leqslant N<2^k-1$，*则一个不成功的查找需要 $k-1$ 次或 k 次比较。* ▌

对二分查找的进一步分析　(对数学没有兴趣的读者，请跳到等式(4)) 树表示也向我们表明了怎样以一种简单的方式来计算平均比较数。令 C_N 是在一次成功的查找中的平均比较数，假定 N 个键码中每一个都是同等可能的变元；并令 C'_N是在一次不成功的查找中的平均比较数，并假定键码之间和端点值外部的 $N+1$ 个区间每一个都是同等可能的。于是由内部和外部路径长度的定义，我们有

$$C_N = 1 + \frac{\text{树的内部路径长度}}{N}$$

$$C'_N = \frac{\text{树的外部路径长度}}{N+1}$$

我们在等式 2.3.4.5-(3)中已经看到，外部路径的长度总是比内部路径长度大 $2N$，因此，在 C_N 和 C'_N之间有一个相当意外的关系

$$C_N = \left(1+\frac{1}{N}\right)C'_N - 1 \tag{2}$$

这个公式是 T.N. Hibbard 给出的 [JACM 9 (1962), 16～17]，它对对应于二叉树的所有查找方法都成立。换言之，它对于所有基于非冗余比较的方法都成立。成功查找比较的方差可用不成功查找比较的方差来表示(见习题 25)。

由上面的公式我们看出，通过比较进行查找的一种“最好的”方式，是在所有具有 N 个内部节点的二叉树中，具有极小外部路径长度的树。幸而，可以证明，在这个意义下，对于所有的 N，算法 B 是最优的；因为我们已经看到(习题 5.3.1－20)，当且仅当一株二叉树的所有外部节点都出现在至多两个相邻的级上时，该二叉树具有极小路径长度。由此得出，对应于算法 B 的树的外部路径长度为

$$(N+1)(\lfloor \lg N\rfloor+2)-2^{\lfloor \lg N\rfloor+1} \tag{3}$$

(见等式 5.3.1-(34))。由这一公式和(2)我们就能计算精确的平均比较数，并假定

所有查找变元都是同等可能的。

$N=$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$C_N=$	1	$1\frac{1}{2}$	$1\frac{2}{3}$	2	$2\frac{1}{5}$	$2\frac{2}{6}$	$2\frac{3}{7}$	$2\frac{5}{8}$	$2\frac{7}{9}$	$2\frac{9}{10}$	3	$3\frac{1}{12}$	$3\frac{2}{13}$	$3\frac{3}{14}$	$3\frac{4}{15}$	$3\frac{6}{16}$
$C'_N=$	1	$1\frac{2}{3}$	2	$2\frac{2}{5}$	$2\frac{4}{6}$	$2\frac{6}{7}$	3	$3\frac{2}{9}$	$3\frac{4}{10}$	$3\frac{6}{11}$	$3\frac{8}{12}$	$3\frac{10}{13}$	$3\frac{12}{14}$	$3\frac{14}{15}$	4	$4\frac{2}{17}$

一般,如果 $k=\lfloor \lg N \rfloor$,则我们有

$$C_N = k+1-(2^{k+1}-k-2)/N = \lg N - 1 + \epsilon + (k+2)/N$$

$$C'_N = k+2-2^{k+1}/(N+1) = \lg(N+1)+\epsilon' \tag{4}$$

其中 $0 \leqslant \epsilon, \epsilon' < 0.0861$,参见等式 5.3.1～(35)。

总结:算法 B 决不做多于$\lfloor \lg N \rfloor + 1$ 次比较,而且它在一次成功的查找中平均进行大约 $\lg N - 1$ 次比较。再没有比这更好的基于比较的查找方法了。如果我们假定这个查找的所有结果都是同等可能的,则程序 B 的平均运行时间近似为

$$\begin{aligned} &(18\lg N - 16)u \quad \text{对于一次成功的查找} \\ &(18\lg N + 12)u \quad \text{对于一次不成功的查找} \end{aligned} \tag{5}$$

一个重要的变形 不使用上述查找中的三个指针 l, i 和 u,而仅使用如下两个量即当前的位置 i 和它的变化速度 δ;在每次不相等的比较之后,我们可以置 $i \leftarrow i \pm \delta$ 和 $\delta \leftarrow \delta/2$(近似地)。这样做是可能的,但正如下面的算法中那样,对其细节需极端小心才成,简单化的解决方法注定会引起失误!

算法 U (均匀的二分查找) 给定一个其键码处于递增次序 $K_1<K_2<\cdots<K_N$ 的记录 $R_1, R_2, \cdots, R_N$ 的表,本算法查找变元 K。如果 N 为偶数,则算法有时将涉及一个虚拟键码 K_0,K_0 应被置成 $-\infty$(或小于 K 的任意值)。我们假定 $N \geqslant 1$。

U1.[初始化] 置 $i \leftarrow \lceil N/2 \rceil$, $m \leftarrow \lfloor N/2 \rfloor$。

U2.[比较] 如果 $K<K_i$,转到 U3;如果 $K>K_i$,转到 U4;如果 $K=K_i$,算法成功地结束。

U3.[i 减值] (我们已经认准包含 m 或 $m-1$ 个记录的一个查找区间;i 恰指向该区间的右端)如果 $m=0$,这个算法以失败告终。否则置$i \leftarrow i - \lceil m/2 \rceil$;然后置$m \leftarrow \lfloor m/2 \rfloor$并返回 U2。

U4.[i 增值](我们已经认准包含 m 或 $m-1$ 个记录的一个查找区间;i 恰指向该区间的右端)如果 $m=0$,这个算法以失败告终。否则置 $i \leftarrow i + \lceil m/2 \rceil$;然后置 $m \leftarrow \lceil m/2 \rceil$并返回 U2。▌

图 6 示出当 $N=10$ 时查找对应的二叉树。当查找失败时,此算法可能在结束前作一次冗余的比较,如图中阴影节点所示。我们可以称这个查找过程为均匀的,因为对于级 l 上的所有节点来说,在级 l 上一个节点的号码与它在$l-1$ 级上的相应

祖宗的号码之差,是一个常数 δ。

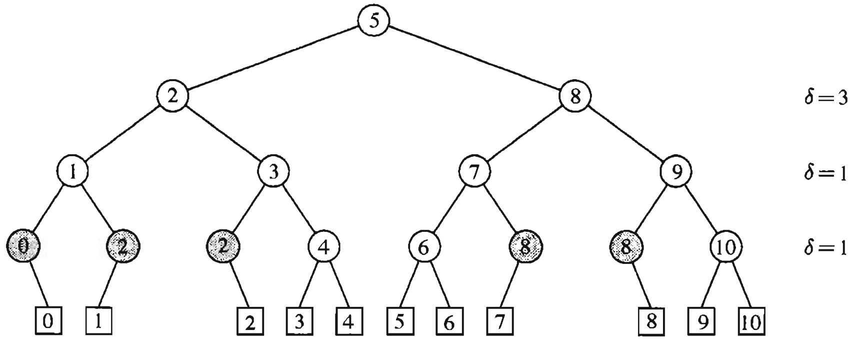


图 6　当 $N=10$ 时一株“均匀的”二分查找树

算法 U 的理论根据可理解如下:假如我们要查找的区间长度为 $n-1$;同中间的元素(n 是偶数)或者同两个中间的元素之一(n 为奇数)进行比较后,剩下了长度为$\lfloor n/2\rfloor-1$ 和$\lceil n/2\rceil-1$ 的两个区间。在重复这一过程 k 次之后,我们得到了 2^k 个区间,其中最小者长度为$\lfloor n/2^k\rfloor-1$,最大者长度为$\lceil n/2^k\rceil-1$。因此,在同一级上两个区间的长度至多差 1;这就使我们有可能选择一个适当的“中间”元素,而无需记住精确的长度。

算法 U 的主要优点是,我们全然不必保持 m 的值,只需要查一张在树的各级中使用的各 δ 值的短表。于是,这个算法简化成下列过程,它在二进计算机上或在十进计算机上同样适用。

算法 C (均匀二分查找)　这个算法与算法 U 很相像,但它用一个辅助表来代替涉及 m 的计算。这个表的项目是

$$\mathrm{DELTA}[j]=\left\lfloor\frac{N+2^{j-1}}{2^j}\right\rfloor\quad 对于\ 1\leqslant j\leqslant\lfloor\lg N\rfloor+2 \tag{6}$$

C1. [初始化]　置 $i\leftarrow$DELTA[1], $j\leftarrow 2$。

C2. [比较]　如果 $K<K_i$,转到 C3;如果 $K>K_i$ 转到 C4;如果 $K=K_i$,此算法成功地结束。

C3. [i 减值]　如果 DELTA[j] = 0,此算法以失败告终。否则,置 $i\leftarrow i-$DELTA[j],$j\leftarrow j+1$,并转到 C2。

C4. [i 增值]　如果 DELTA[j] = 0,此算法以失败告终。否则,置 $i\leftarrow i+$DELTA[j],$j\leftarrow j+1$,并转到 C2。▌

习题 8 证明,这个算法仅当 N 是偶数时才访问人为的键码 $K_0=-\infty$。

程序 C (均匀二分查找)　这个程序以 rA≡K,rI1≡i,rI2≡j,rI3≡DELTA[j],利用算法 C,所作的事和程序 B 相同。

```
START   ENT1   N+1/2        1      C1.初始化 i←⌊(N+1)/2⌋
        ENT2   2            1      j←2
```

```
              LDA    K            1
              JMP    2F           1
    3H        JE     SUCCESS      C1          如果 K=K_i 则转移
              J3Z    FAILURE      C1-S        如果 DELTA[j]=0,则转移
              DEC1   0,3          C1-S-A      C3. i 减值
    5H        INC2   1            C-1         j←j+1
    2H        LD3    DELTA,2      C           C2. 比较
              CMPA   KEY,1        C
              JLE    3B           C           如果 K≤K_i 则转移
              INC1   0,3          C2          C4. i 增值
              J3NZ   5B           C2          如果 DELTA[j]≠0 则转移
    FAILURE   EQU    *            1-S         如果不在表中则转出  ▌
```

在一次成功的查找中,这个算法对应的二叉树与算法 B 的二叉树有相同的内部路径长度,所以平均比较次数 C_N 和以前一样。在一次不成功的查找中,算法 C 总是恰好进行$\lfloor \lg N \rfloor + 1$ 次比较。程序 C 的总运行时间在左、右分支之间不是十分对称的,因为 $C1$ 被赋予的权比 $C2$ 要大。但习题 11 表明,$K < K_i$ 同$K > K_i$ 的机会大体上相同;因此程序 C 近似地花费

$$\begin{aligned}&(8.5 \lg N - 6)u \quad \text{对于一次成功的查找}\\&(8.5\lfloor \lg N \rfloor + 12)u \quad \text{对于一次不成功的查找}\end{aligned} \tag{7}$$

尽管对于程序 B 的运行时间(5)还假定 MIX 有一条“右移二进位”指令,如果不使用二进计算机的任何特殊性质的话,这个速度相当于程序 B 的两倍多。

L.E. Shar 于 1971 年提议对二分查找作另一种修改,这在某些计算机上实现起来会更快一些,因为在第一步之后它是均匀的,而且它不需要表。第一步是比较 K 和 K_i,其中,$i = 2^k$,$k = \lfloor \lg N \rfloor$。如果 $K < K_i$,则对于 δ 等于 $2^{k-1}, 2^{k-2}, \cdots, 1, 0$ 使用一个均匀查找。另一方面,如果 $K > K_i$,则重置 i 为 $i' = N + 1 - 2^l$,其中,$l = \lceil \lg(N - 2^{k+1}) \rceil$,并假设头一个比较实际上是 $K > K_{i'}$,且对于 δ 等于 $2^{l-1}, 2^{l-2}, \cdots, 1, 0$ 使用一个均匀查找。

图 7 示出了当 N = 10 时 Shar 的方法。和以前的算法一样,它的比较次数决不会超过$\lfloor \lg N \rfloor + 1$;因此,尽管有时接连通过的某些步骤是多余的,但其比较数至多比极小平均比较次数多一次(参见习题 12)。

二分查找还有另一种修正,当 N 非常大时,它提高了所有上述方法的速度,这个方法在习题 23 中讨论,更快的方法见习题 24。

***斐波那契查找** 我们已经看到,在多阶段合并中,斐波那契数可以起到和 2 的乘方相类似的作用。在查找中也出现类似的现象,在这里斐波那契数为我们提供了二分查找的另一个方案。从而得到某些计算机上更可取的方法,因为它仅包含加法

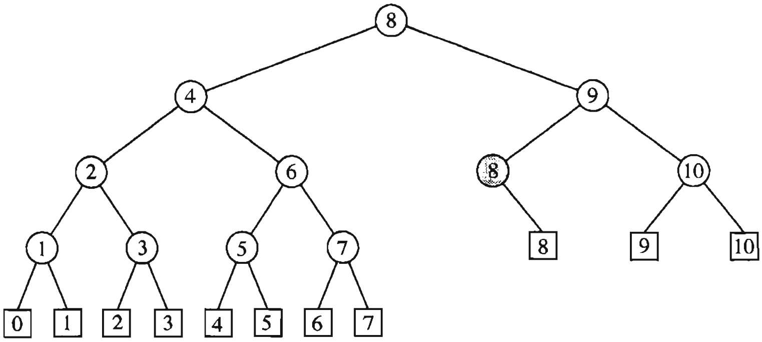


图 7　当 $N=10$ 时,Shar 的准均匀二叉树

和减法,没有除以 2 的除法。我们所要讨论的过程,应同一个确定单峰函数极大值位置称为“斐波那契查找”的重要数值过程区别开来[参见*Fibonacci Quarterly* **4** (1966),265～269];名称的相似性已经导致了一些混乱。

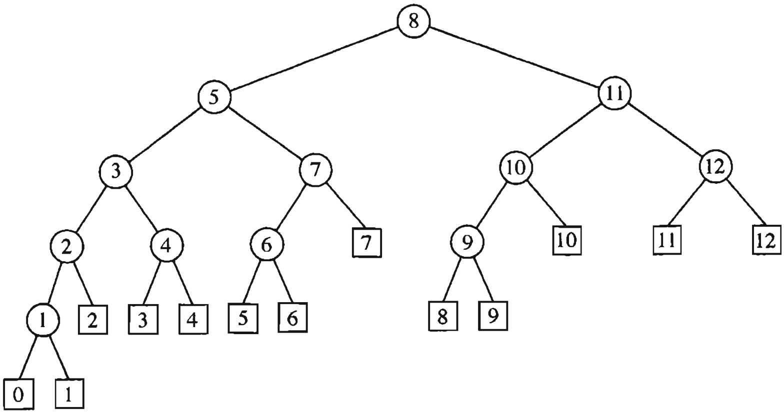


图 8　6 阶斐波那契树

乍一看,斐波那契查找技术似乎非常神秘,如果我们简单地把程序拿来并试图看看它在干些什么的话,它似乎是在变魔术。但只要把对应的查找树画出来,神秘感立即就消失了。因此,我们将通过考察斐波那契树来开始对这个方法进行研究。

图 8 示出了 6 阶的斐波那契树。与我们已经考虑过的其它树相比,它看起来更像现实生活中的灌木,这也许是因为许多自然过程都满足一种斐波那契规律的缘故。一般,阶数 k 的斐波那契树有 $F_{k+1}-1$ 个内部(圆形)节点和 F_{k+1}个外部(正方形)节点,这株树可构造如下:

如果 $k=0$ 或 $k=1$,则此树就是$\boxed{0}$。

如果 $k\geqslant 2$,则树根为 F_k;左子树是阶数为 $k-1$ 的斐波那契树;右子树是阶数为 $k-2$ 且所有编号都增加 F_k 的斐波那契树。

同一个数,这个数就是一个斐波那契数。例如,在图 8 中,$5=8-F_4$ 和 $11=8+F_4$。当差是 F_j 时,左下分支对应的斐波那契差数是 F_{j-1},而右下分支的差数降为 F_{j-2}。例如,$3=5-F_3$,而 $10=11-F_2$。

如果我们把这些观察同识别外部节点的一个适当的机制结合起来,就得到下列方法。

算法 F (斐波那契查找) 给定一个其键码处于递增次序 $K_1<K_2<\cdots<K_N$ 的记录 $R_1,R_2,\cdots,R_N$ 的表,本算法查找一个给定的变元 K。

为了叙述方便,假定 $N+1$ 是一个完全的斐波那契数 F_{k+1}。如果提供适当的初始化(见习题 14),就不难使这个方法对于任何 N 都有效。

F1.[初始化] 置 $i\leftarrow F_k$,$p\leftarrow F_{k-1}$,$q\leftarrow F_{k-2}$。(在整个算法中,p 和 q 总是相继的斐波那契数)。

F2.[比较] 如果 $K<K_i$,则转到步骤 F3;如果 $K>K_i$,则转到 F4;而如果 $K=K_i$,则算法成功地结束。

F3.[i 减值] 如果 $q=0$,则算法以失败告终。否则置 $i\leftarrow i-q$,并置 $(p,q)\leftarrow(q,p-q)$,然后返回 F2。

F4.[i 增值] 如果 $p=1$,则算法以失败告终,否则置 $i\leftarrow i+q$,$p\leftarrow p-q$,然后 $q\leftarrow q-p$,并返回 F2。 ▌

以下的 MIX 实现,通过设置两个副本的内循环而赢得了速度,一个内循环把 p 放在 rI2 中,q 放在 rI3 中,而另一个则恰好把这两个寄存器颠倒过来;这便简化了步骤 F3。事实上,程序在寄存器中真正保留的是 $p-1$ 和 $q-1$,而不是 p 和 q,为的是简化步骤 F4 的判断"$p=1$?"。

程序 F (斐波那契查找) 我们遵循前面的约定 $rA\equiv K$,$rI1\equiv i$,(rI2 或 rI3)$\equiv p-1$,(rI3 或 rI2)$\equiv q-1$。

START	LDA	K	1	F1. 初始化
	ENT1	F_k	1	$i\leftarrow F_k$
	ENT2	$F_{k-1}-1$	1	$p\leftarrow F_{k-1}$
	ENT3	$F_{k-2}-1$	1	$q\leftarrow F_{k-2}$
	JMP	$F2A$	1	转到步骤 F2
F4A	INC1	1,3	$C2-S-A$	F4. i 增值。$i\leftarrow i+q$
	DEC2	1,3	$C2-S-A$	$p\leftarrow p-q$
	DEC3	1,2	$C2-S-A$	$q\leftarrow q-p$
F2A	CMPA	KEY,1	C	F2. 比较
	JL	F3A	C	如果 $K<K_i$ 则转到 F3
	JE	SUCCESS	$C2$	如果 $K=K_i$ 则转出
	J2NZ	F4A	$C2-S$	如果 $p\neq1$ 则转到 F4

12		J2NZ	F4A	$C2-S$	如果 $p\neq 1$ 则转到 F4
13		JMP	FAILURE	A	如果不在表中则转出
14	F3A	DEC1	1,3	$C1$	<u>*F3. i 减值*</u>。$i\leftarrow i-q$
15		DEC2	1,3	$C1$	$p\leftarrow p-q$
16		J3NN	F2B	$C1$	如果 $q>0$ 则交换寄存器
17		JMP	FAILURE	$1-S-A$	如果不在表中则转出
18	F4B	INC1	1,2		(18～29 行与 06～17 行并行)
19		DEC3	1,2		
20		DEC2	1,3		
21	F2B	CMPA	KEY,1		
22		JL	F3B		
23		JE	SUCCESS		
24		J3NZ	F4B		
25		JMP	FAILURE		
26	F3B	DEC1	1,2		
27		DEC3	1,2		
28		J2NN	F2A		
29		JMP	FAILURE		▎

习题 18 中分析了这个程序的运行时间。图 8 表明，而且分析证明，通常取左分支的机会比右分支稍多些。设 C, $C1$ 和 $(C2-S)$ 分别为步骤 F2、F3 和 F4 被执行的次数，则我们有

$$
\begin{aligned}
C &= (\text{ave } \phi k/\sqrt{5}+O(1),\ \max k-1)\\
C1 &= (\text{ave } k/\sqrt{5}+O(1),\ \max k-1) \qquad (8)\\
C2-S &= (\text{ave } \phi^{-1}k/\sqrt{5}+O(1),\ \max\lfloor k/2\rfloor)
\end{aligned}
$$

于是取左分支的次数大约是取右分支次数的 $\phi=1.618$ 倍（这是我们能够猜想得到的一个事实，因为每个探测都把剩下的区间分成为两部分，且左边部分大约是右边部分的 ϕ 倍那么大）。因此对于一次成功的查找，程序 F 总的平均运行时间近似于

$$\frac{1}{5}((18+4\phi)k+31-26\phi)u \approx (7.050\lg N+1.08)u \qquad (9)$$

对于一次不成功的查找，加上 $(9-3\phi)u\approx 4.15u$。尽管最坏情况下运行时间（大约为 $8.6\lg N$）稍微慢些，但它比程序 C 更快些。

内插查找　让我们暂时忘记计算机，来考虑人们如何真正地进行一次查找。有时日常生活会向我们提示好的算法。

想像你自己在一部字典中查一个字，你大概不是从查中间的页开始，然后查 1/4

或3/4处的页,等等,如同在二分查找中那样。你更不大可能去使用斐波那契查找!

如果你所要的字是以字母 A 开始的,那你或许会从靠近字典前头的地方开始。事实上,许多字典都有“书边索引”,它标出以一个固定字母开始的单字的起始页或中间页。这个书边索引技术不难用到计算机上去,而且它将加速查找,6.3 节中剖析了这些算法。

即使在找到了查找的起始点之后,你的动作也还不太像我们已经讨论过的方法。如果你注意到,所要找的字按字母次序比你正在查找的页上的字大很多,则在你开始进行下次查找之前,就要翻过好多页。这和上面的一些算法大不相同,那些算法并没有对“大得多”和“稍微大些”加以区别。

这些考虑提示了一个堪称为*内插查找*的算法:当知道 K 位于 K_l 和 K_u 之间时,我们可以把下一次探测的位置选在位于 l 和 u 之间大约 $(K-K_l)/(K_u-K_l)$ 这一点上,这里假定键码是数值的,而且键码在整个这一区间里以大体不变的方式增值。

内插查找渐近地优于二分查找。当表中的键码是随机分布时,则二分查找的每一步把查找工作量从 n 降低到 $\frac{1}{2}n$,而内插查找则把 n 降到 $\sqrt{n}$。因此,平均说来,内插查找花费 $\lg\lg N$ 步,把不确定性从 N 减少到 2。

然而,计算机的模拟实验表明,除非表相,不然内插查找并不把比较数减少到足以补偿所需的多余计算的程度。除非 N 超过比如说 $2^{16}=65536$。不然典型的文件并不充分的随机,而且 $\lg\lg N$ 和 $\lg N$ 之间差别并不很大。在查找可能很大的外部文件的早期阶段时内插最为成功;在这个范围收窄之后,二分查找能更快速地完成任务。(注意,用手翻阅词典实质上是一个外部查找,而非一个内部查找,稍后我们将讨论外部查找)。

历史和文献 已知最早的把一长串项目排成顺序以便于查找的例子,是大约公元前 200 年就编制成的、值得注意的巴比伦人的艾娜基比特-安奴(Inakibit-Anu)倒数表。这个陶土做的表包含 100 对以上的值,它们看来是大约 500 个多精度的六十进制的数和它们的倒数的一个表的开始,并且已按词典顺序排序。例如,该表包括有下列项的序列:

01 13 09 34 29 08 08 53 20	49 12 27
01 13 14 31 52 30	49 09 07 12
01 13 43 40 48	48 49 41 15
01 13 48 40 30	48 46 22 59 25 25 55 33 20
01 14 04 26 40	48 36

对这样 500 个项进行排序的任务,在当时可利用的技术下,确实是非凡的[关于进一步的细节,参见 D. E. Knuth, *Selected Papers on Computer Science* (Cambridge Univ. Press, 1996),第 11 章]。

把数值排成次序是相当自然的,但是字母或字之间的次序关系并不是那样容易看出的。然而对于单个字母的一个整理好的序列已经出现在最古老的字母表中。例如,许多圣经赞美诗都有这样的诗句,它遵循一个严格的字母序列,头一个诗句以

a 开始，第二句以 b 开始，等等；这有助于记忆。终于，字母的标准序列为闪米特人(Semitic)和希腊人用来表示数；例如，α，β，γ 分别表示 1，2，3。

把字母表的次序用于整个的字，是一项更晚些的发明；显而易见，但必须教给孩子们，而且在历史上的某个时候，曾有必要教给成年人！在爱琴岛上已经发现了大约公元 300 年前的若干表，给出了某些迷信的宗教中的人名；这些表按字母排列，但仅按头一个字母，因此只表示了自左至右进行基数排序的第一次扫描。公元 134～135 年间的某些希腊文稿包含了一些分类账的片断，它列出了按头两个字母排序的纳税人的名称。(Apollonius Sophista)在他的长而和谐的荷马诗中(公元 100 年)，把字母顺序用于头两个字母上，而且通常也用于随后的字母上。还有一些更完美的字母顺序化的例子非常有名，最突出的有盖伦 Galen 的*Hippocratic Glosses*（大约公元 200 年)，但这些都是非常罕见的。因此，在圣人 St. Isidorus 的*Etymologiarum*（大约公元 630 年，第十册)中，仅按它们的头一个字母来排列词序；而*Corpus Glossary*（约公元 725 年)仅使用每个字的头两个字母。后两部著作也许是中世纪编纂的最大的非数值方面的数据文件。

在 Giovanni di Genoa 的*Catholicon*（1286 年)一书之前，我们还没有见到过对正确的字母顺序的专门描述。在他的前言中，Giovanni 说明：

amo 在　bibo　之前
abeo 在　adeo　之前
amatus 在　amor　之前
imprudens 在　impudens　之前
iusticia 在　iustus　之前
polisintheton 在　polissenus　之前

(由此给出了按第一个，第二个，…，第六个字母排列的情况的例子)，其余类推。他述说了为想出这些规则所需要的努力。“因此，好读者，我请求你，不要讥笑我这巨大的劳动和这个次序是不值钱的。”

在发明印刷术之前，Lloyd W. Daly 对字母顺序进行过详细研究[见*Collection Latomus* 90 (1967)，100 页]，他发现了某些有趣的旧手稿，它们显然是一些草稿纸，上面有按头几个字母排序的一些单字(见该专著的 87～90 页)。

第一本英文字典，即 Robert Cawdrey 的*Table Alphabeticall*（伦敦，1604)]包含下列的提示：

如果你想要查找的字是以(a)开始的，则在这个表的开头处查找，如果是以(v)开始，则在靠近这个表的末尾处查找。又，如果字以(ca)开始，则在字母(c)的开头处查找，如果以(cu)开始，则在靠近该字母的末尾处查找，如此类推。

Gawdrey 在编写他的字典的过程中，看来也曾自学过怎样来编排字母；在最初的一些页上，出现过许多放错位置的字，而最后部分的字母顺序就好得多了。

John Mauchly 在也许是第一部出版的关于非数值程序设计方法的书中，首先提出了二分查找[*Theory and Techniques for the Design of Electronic Digital Computers*，G. W. Patterson 主编，**1**(1946)，9.7～9.8；3 (1946)，22.8～22.9]。这个方法在程序

员中“众所周知”，但是似乎还没有人对 N 不具有 2^n-1 这种特殊形式时的方法作过详细讨论[见 A. D. Booth, *Nature* **176**(1955), 565; A. I. Dumey, *Computers and Automation* **5**(1956 年 12 月), 7, 其中二分查找称为“第二十个问题”; Daniel D. McCracken, *Digital Computer Programming* (Wiley, 1957), 201～203; 以及 M. Halpern *CACM* **1**, 1(1958 年 2 月), 1～3]。

D. H. Lehmer[*Proc. Symp. Appl. Math.* **10**(1960), 180～181]显然是第一个发表对于所有 N 都有效的一个二分查找算法的人，H. Bottenbruch 迈出了第二步[*JACM* **9** (1962), 214]，他介绍了算法 B 的一种有趣的变形，它避免了在最后结束之前单作一次相等判断：在步骤 B2 中利用 $i \leftarrow \lceil (l+u)/2 \rceil$ 代替 $\lfloor (l+u)/2 \rfloor$，每当 $K \geqslant K_i$ 时置 $l \leftarrow i$；然后在每一步中 $u-l$ 都减值。最后，当 $l=u$ 时，我们有 $K_l \leqslant K \leqslant K_{l+1}$，而且再做一次比较即可判断这个查找是否成功(假定开始时 $K \geqslant K_1$)。这个思想在许多计算机上稍微加速了内循环，而且同样的原理可以在这一节中讨论的所有算法中使用；但是由于式(2)，一次成功的查找平均将要求大约再做一次迭代。由于内循环仅执行大约 lg N 次，因此在这一次额外的迭代与一次更快的循环之间的折衷并不节省时间，除非 N 非常大(见习题 23)。另一方面，当表含重复键码时，Bottenbruch 的算法将发现一个给定键码的最右出现，而这一性质有时很重要。

K. E. Iverson[*A Programming Language* (Wiley, 1962), 141]给出了算法 B 的过程，但是没有考虑到不成功查找的可能性。D. E. Knuth[*CACM* 6(1963), 556～558]介绍了算法 B 作为自动画框图系统所用的一个例子。均匀二分查找算法 C，是 1971 年斯坦福大学的 A. K. Chandra 向作者建议的。

斐波那契查找是由 David. E. Ferguson[*CACM* 3(1960), 648]发明的。类似于斐波那契树的二叉树早在 1910 年就出现在挪威数学家(Axel Thue)先驱性的工作中。(参见习题 28)。没有标号的斐波那契树早在许多年前就出现了，它作为珍品收藏在 Hugo Steinhaus 的普及读物 *Mathematical Snapshots* [纽约：Stechert, 1938]第 28 页；他把这棵树倒着画，因而看起来像一颗真实的树，树的右分支的长度是左分支的两倍，使得所有叶子都在同一级上出现。

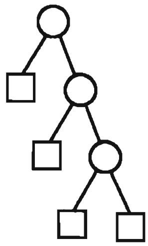
W. W. Peterson 提出了内插查找[*IBM J. Res, & Devel.* **1**(1957), 131～132]；关于它的平均特性的正确分析直到许多年后才被发现。

习 题

▶1. [*21*] 证明：如果在二分查找的步骤 B2 中 $u<l$，则我们有 $u=l-1$ 和 $K_u<K<K_l$(由约定，假定 $K_0=-\infty$ 和 $K_{N+1}=+\infty$，但这些人为的键码在本算法中是用不上的，所以它们不必在实际的表中出现)。

▶2. [*22*] 如果我们(a)把步骤 B5 改为“$l \leftarrow i$”来代替“$l \leftarrow i+1$”，则算法 B 是否仍将有效地工作？(b)把步骤 B4 改为“$u \leftarrow i$”以代替“$u \leftarrow i-1$”呢？(c)这两个变化都作呢？

3. [*15*] 什么查找方法对应于下页上边的树？什么是在一次成功的查找中所作的平均比较次数？在一次不成功的查找中呢？



4.[20] 如果使用程序 6.1S(顺序查找),一次查找恰花费 638 个单位时间,则使用程序 B(二分查找)须花费多长时间?

5.[M24] 对于 N 的什么值,程序 B 实际上平均慢于一个顺序查找(程序 6.1Q′)(假定这个查找是成功的)?

6.[28] (E.K.Iverson) 习题 5 建议我们最好采用一种"混合"方法,即当剩下区间的长度小于某个谨慎地选择的值时,就从二分查找变成顺序查找。试对这样一个查找写出有效的 MIX 程序并确定最适于"换马"的值。

▶7.[M22] 如果我们改变步骤 U1 使得:(a) i 和 m 都置成等于 $\lfloor N/2\rfloor$;(b) i 和 m 都置成等于 $\lceil N/2\rceil$,试问算法 U 是否仍将正确地工作?[提示:假设头一步是"置 $i\leftarrow 0$, $m\leftarrow N$(或 $N+1$),转到 U4"]?

8.[M20] 像在(6)中定义的那样,令 $\delta_j=\mathrm{DELTA}[j]$是在算法 C 中的第 j 个增量。

a)和 $\sum_{j=0}^{\lfloor\lg N\rfloor+2}\delta_j$ 等于什么?

b)在步骤 C2 中能出现的 I 的极小值和极大值是多少?

9.[20] 对于所有的查找变量,算法 B 和算法 C 实现相同的比较序列,在这个意义下,算法 B 和 C 精确地等价,有无任何 $N>1$ 的值,使算法 B 和 C 精确地等价?

10.[21] 说明如何来写包含有近似地 $7\lg N$ 条指令和有大约 $4.5\lg N$ 单位的运行时间的算法 C 的 MIX 程序。

11.[M26] 作为 N 和 S 的一个函数,求程序 C 的频率分析中 $C1$、$C2$ 和 A 的平均值的精确公式。

12.[20] 画出对应于 $N=12$ 时 Shar 方法的二分查找树。

13.[M24] 对于 $1\leqslant N\leqslant 16$,试编出 Shar 方法所作的平均比较次数表,并且既考虑成功的查找也考虑不成功的查找。

14.[21] 说明怎样推广算法 F,使它对所有 $N\geqslant 1$ 都将适用。

15.[M19] 对于 k 的什么值,k 阶斐波那契树定义了一个最优查找过程,即平均说来所做的比较次数最少?

16.[21] 图 9 示出在斐波那契原来的兔子问题中(参考 1.2.8 节)兔子的线性图表。问在这个图表和正文中讨论的斐波那契树之间是否有一个简单的关系?

17.[M21] 由习题 1.2.8－34(或习题 5.4.2－10)我们知道,每个正整数 n 均可惟一地表示为斐波那契数之和 $n=F_{a_1}+F_{a_2}+\cdots+F_{a_r}$,其中 $r\geqslant 1$,对于 $1\leqslant j<r$, $a_j\geqslant a_{j+1}+2$,且 $a_r\geqslant 2$。试证明在 k 阶斐波那契树中,由根到节点ⓝ 的路径长度为 $k+1-r-a_r$。

18.[M30] 作为 k、F_k、F_{k+1}和 S 的一个函数,求在程序 F 的频率分析中 $C1$、$C2$ 和 A 的平均值的精确公式。

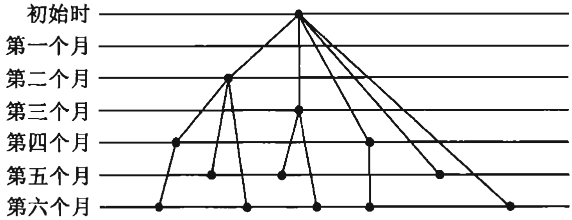


图 9 按照斐波那契规则繁殖的兔子对

19.[*M42*] 对于习题 14 中建议的算法的平均运行时间进行详尽的分析。

20.[*M22*] 在一个二分查找中需要的比较次数近似为 $\log_2 N$,而在斐波那契查找中它大约为 $(\phi/\sqrt{5})\log_\phi N$。本题的目的是证明这些公式都是一个更一般结果的特殊情况。

设 p 和 q 是正数,且 $p+q=1$。考虑一个查找算法,对于它,给定递增的 N 个数的一份表,首先把变元同第(pN)个键码比较,然后在诸较小的块区中迭代这个过程$\left(\text{二分查找有 } p=q=\frac{1}{2}\text{;斐波那契查找有 } p=1/\phi, q=1/\phi^2\right)$。

如果 $C(N)$ 表示查找长度为 N 的一份表所需要的平均比较次数,则它近似地满足关系式

$$C(1)=0;\ C(N)=1+pC(pN)+qC(qN)\quad \text{对于 } N>1$$

这是由于在头一次比较之后,该查找归结为一个 pN 个元素的查找的概率(大约)为 p,归结为一个 qN 个元素的查找的概率为 q 的缘故。当 N 很大时,我们可以忽略由于 pN 和 qN 不恰为整数这一事实所引起的小阶次的影响。

a)证明,适当选择 b,$C(N)=\log_b N$ 恰满足这些关系,对于二分和斐波那契查找,b 的这个值同早先导出的相一致。

b)考虑以下的论点:"在这个算法中,被扫描的区间长度除以 $1/p$ 的概率是 p;这个区间大小除以 $1/q$ 的概率是 q。因此平均说来该区间除以 $p\cdot(1/p)+q\cdot(1/q)=2$,所以这个算法和二分查找一样好,而不论 p 和 q 是什么。"这个论证有错误吗?

21.[*20*] 画出对应于 $N=10$ 时内插查找的二叉树。

22.[*M41*](姚期智和姚储枫)证明:当把内插查找应用于已经排好序的 N 个独立的均匀随机键码时,平均说来,它的一个适当陈述在渐近意义下要求平均 $\lg\lg N$ 次比较。而且,对于这样的表的所有查找算法平均必须花费近似 $\lg\lg N$ 次比较。

▶23.[*25*] 在本节末尾提出的 H. Bottenbruch 的二分查找算法,在查找的末尾之前避免作相等判断。(在该算法运行期间,我们知道 $K_l \leqslant K < K_{u+1}$,而且在 $l=u$ 之前,不检查相等的情形)。这样一个技巧将使程序 B 对大的 N 运行得更快一些,因为"JE"指令可从内循环中撤销。(然而,这一思想事实上并不实际,因为 $\lg N$ 通常很小;为了补偿在一次成功的查找时所需要的额外的工作,要 $N>2^{66}$才行,因为(5)的运行时间$(18\lg N-16)u$ 被减少到$(17.5\lg N+17)u$!)

证明对应于一个二叉树的每个查找算法,均可修改为如下一个查找算法,这个查找算法在树的内部节点处使用两路分支(<和≥),代替在正文讨论中所用的三路分支(<,=,或>)。特别是,说明在这种情况下如何修改算法 C。

▶24.[*23*] 在 2.3.4.5 小节和 5.2.3 小节已经看到,完备的二叉树是一种在连续单元中表示一株极小路径长度树的方便的方式。试设计以这个表示为基础的一个有效查找方法。[提示:在二分查找中是否有可能用乘以 2 来代替除以 2?]

▶25.[*M25*] 假设一株二叉树对于 $k=0,1,\cdots$ 在 k 级上有 a_k 个内部节点和 b_k 个外部节点(根

在 0 级)，于是在图 8 中我们有$(a_0,a_1,\cdots,a_5)=(1,2,4,4,1,0)$和$(b_0,b_1,\cdots,b_5)=(0,0,0,4,7,2)$。(a)证明在生成函数 $A(z)=\sum_k a_k z^k$ 和$B(z)=\sum_k b_k z^k$ 之间有一个简单的代数关系成立。(b)一株二叉树的成功查找的概率分布，具有生成函数 $g(z)=zA(z)/N$；而不成功的查找的生成函数是 $h(z)=B(z)/(N+1)$。(于是在正文的记号下，我们有 $C_N=\text{mean}(g)$，$C'_N=\text{mean}(h)$，且等式(2)给出了这些量之间的一个关系式。) 试求 $\text{var}(g)$和 $\text{var}(h)$之间的一个关系式。

26.[*22*] 证明斐波那契树同三条磁带上的多阶段合并排序有关。

27.[*M30*] (H.S.Stone 和 John Linn)考虑一个查找过程，它同时使用 k 部处理机，且仅以键码的比较为基础。于是在查找的每一步，确定 k 个下标 $i_1,\cdots,i_k$，同时进行 k 个比较；如果对于某个 j，$K=K_{i_j}$，则该查找成功地结束，否则，根据 2^k 种可能的结果 $K<K_{i_j}$ 或 $K>K_{i_j}$，$(1\leqslant j\leqslant k)$来进行下一步骤。

证明，当 $N\to\infty$时，这样一个过程必须总是近似地至少平均花费 $\log_{k+1}N$ 步，其中假定这个表的每个键码作为一个查找变元是同等可能的。(因此，比起一部处理机的二分查找来，在速度上可能的增加仅是因子 $\lg(k+1)$，而不是我们可能期望的因子 k。在这个意义下，更有效的是对每个处理机赋以不同的独立的查找问题而不是把它们在单一的查找中合并。)

28.[*M23*] 借助于在二元操作符 $*$ 的代数表达式定义 Thue 树 T_n 如下：$T_0(x)=x*x$，$T_1(x)=x$，$T_{n+2}(x)=T_{n+1}(x)*T_n(x)$。

a) T_n 的叶数是当把 $T_n(x)$完全写出时 x 出现的个数，试借助斐波那契数来表达这个数。

b)证明，如果二元操作符 $*$ 满足公理

$$((x*x)*x)*((x*x)*x)=x$$

则对于所有 $m\geqslant 0$ 和 $n\geqslant 1$，$T_m(T_n(x))=T_{m+n-1}(x)$

▶29.[*22*] (Paul Feldman，1975) 代替假定 $K_1<K_2<\cdots K_N$，仅假定 $K_{p(1)}<K_{p(2)}<\cdots<K_{p(N)}$，其中排列 $p(1)p(2)\cdots p(N)$是一个乘方，而且对于 j 的所有偶数值，$p(j)=j$。试证明，通过作到至多 $2\lfloor\lg N\rfloor+1$ 次比较，我们能够找出任何给定的键码 K，或者确定 K 不存在。

30.[*27*] (乘方的编码)使用上一道题的思想，试找出一种方法，以这样一种方式来安排 N 个不同的键码，即当 $m\leqslant N/4+1-2^t$ 时，这些键码的相对次序含蓄地对任意给定的 t 位二进数 $x_1,x_2,\cdots,x_m$ 的一个数组进行编码。通过你的安排，对于任何给定的 j，只须作 k 次比较，应有可能确定 x_j 的前导的k 个二进位，也有可能通过$\leqslant 2\lfloor\lg N\rfloor+1$ 次比较找出任意一个键码。(这个结果可用于对在时间和空间两方面都是渐近地有效的一些数据结构的理论研究上。

6.2.2　二叉树查找

由上一小节我们知道，一株隐含的二叉树结构有助于了解二分查找和斐波那契查找的特性。对于给定的 N 值，与二分查找相对应的树，达到了借助键码比较来查找一份表所需比较数的理论极小值。但是，上一小节的方法主要适用于固定长度的表，这是因为，记录的顺序分配使得插入和删除相当费时。如果这个表动态地变化，则我们花费的维护时间可能比在二分查找中查找它们时节省的时间还多。

使用一个明显的二叉树结构，不但能有效地查找表，而且还能迅速地插入和删除记录。因此，我们实际上就有一个对于查找和排序两者都有用的方法。这种灵活性的获得，是通过对表的每个记录附加两个链接场来达到的。

查找一个增长着的表的技术，通常称为*符号表算法*，因为汇编程序和编译程序

以及其它系统程序一般都使用这样的方法以记住用户定义过的符号。例如,在一个编译程序内每个记录的键码,可以是某个 FORTRAN 或 C 程序中表示某变量的符号标识符,而该记录的其余部分可以包含那个变量的类型及它的存储分配的信息。或者说,键码可以是 MIXAL 程序中的一个符号,而该记录的其余部分包含着那个符号的等价物。本小节描述的树查找和插入程序可有效地用作符号表算法,特别适用于一些希望按字母顺序打印出符号表的场合。6.3 和 6.4 节描述了其它符号表算法。

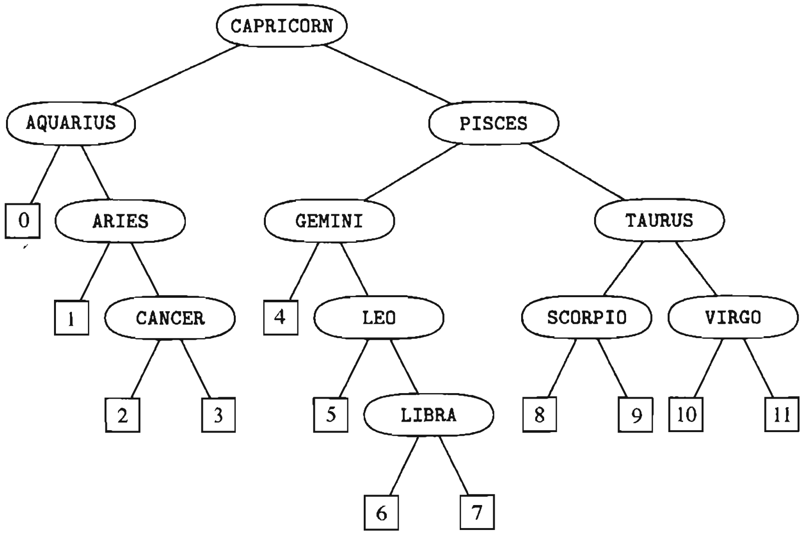


图 10 一株二分查找树

图 10 示出包含黄道十一天宫名称的二分查找树。如果我们现在在根处或树的顶点处开始查找第 12 个名字人马座(SAGITTARIUS),则发现它大于摩羯座(CAPRICORN),所以我们向右移;它大于双鱼座(PISCES),所以再次右移;它小于金牛座(TAURUS),所以左移;它还小于天蝎座(SCORPIO),所以我们达到外部节点[8]。这个查找是不成功的;现在我们可以把人马座链接到树中以代替外部节点[8],这就把它插入到了最后的查找位置。这样一来,这个表就能增长而无需移动任何现有的记录。图 10 就是以一株空树开始,把 CAPRICORN,AQUARIUS,PISCES,ARIES,TAURUS,GEMINI,CANCER,LEO,VIRGO,LIBRA,SCORPIO 诸键码依次插入树中而形成的。

图 10 中根的左子树的所有键码,在字符次序上都小于 CAPRICORN,而右边子树中的所有键码,在字符次序上都大于它。对于每个节点的左子树和右子树,类似的命题也成立。由此得出,如果我们以对称次序来遍历这株树(参看 2.3.1 小节),那么诸键码严格地从左到右组成如下序列

AQUARIUS, ARIES, CANCER, CAPRICORN, GEMINI, LEO, …, VIRGO

这是因为,对称次序的原则是:对每个节点,先遍历该节点的左子树,接着是该节点

本身,然后遍历右子树。

下列算法详尽地叙述了查找和插入的过程。

算法 T(树查找和插入)　给定一个形式如上所述二叉树的记录表,本算法查找一个给定的变元 K。如果 K 不在表中,则在树的适当位置插入包含 K 的一个新节点。

假定树的节点至少包含下列诸字段

KEY(P) = 存于 NODE(P) 中的键码;
LLINK(P) = 指向 NODE(P) 的左子树的指针;
RLINK(P) = 指向 NODE(P) 的右子树的指针。

空子树(图 10 中的外部节点)以空指针 Λ 表示,变量 ROOT 指向此树的根。为方便起见,我们假定此树不是空的(即,ROOT≠Λ)。因为当 ROOT = Λ 时,应该做什么操作是显然的。

T1.[初始化]　置 P←ROOT。(指针变量 P 将沿树下移)。

T2.[比较]　如果 K<KEY(P),则转到 T3;如果 K>KEY(P),则转到 T4;如果 K = KEY(P),则查找成功地结束。

T3.[左移]　如果 LLINK(P)≠Λ,则置 P←LLINK(P)并转回 T2,否则转到 T5。

T4.[右移]　如果 RLINK(P)≠Λ,则置 P←RLINK(P)并转回 T2。

T5.[插入树](这个查找是不成功的;我们现在将把 K 置入树中。)　置 Q⇐AVAIL,即一个新节点的地址。置 KEY(Q)←K,LLINK(Q)←RLINK(Q)←Λ (实际上,新节点的其它场也应予初始化。)如果 K 小于 KEY(P),则置 LLINK(P)←Q,否则置 RLINK(P)←Q。(这时我们可置 P←Q,并成功地结束此算法)。▌

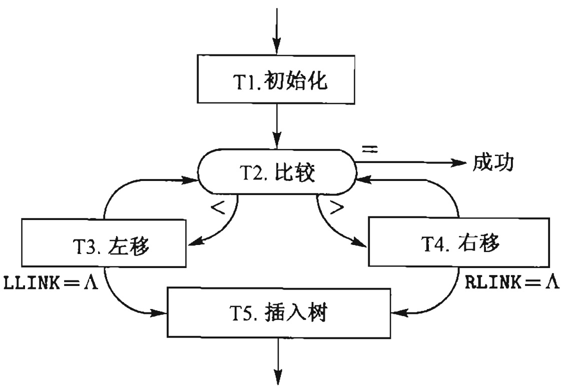


图 11　树查找和插入

这个算法很便于用机器语言实现。例如,我们可以假定,树节点形为

+	0	LLINK	RLINK
		KEY	

(1)

也许后边还跟有 INFO 的一些附加字。利用可用存储空间的一个 AVAIL 表,如同在第二章中那样,我们可以写出下列的 MIX 程序:

程序 T (树查找和插入) rA≡K,rI1≡P,rI2 = Q。

```
01  LLINK  EQU   2:3
02  RLINK  EQU   4:5
03  START  LDA   K              1          T1.初始化
04         LD1   ROOT           1          p←ROOT
05         JMP   2F             1
06  4H     LD2   0,1 (RLINK)    C2         T4.右移,Q←RLINK(P)
07         J2Z   5F             C2         如果 Q = Λ  则转到 T5
08  1H     ENT1  0,2            C - 1      P←Q
09  2H     CMPA  1,1            C          T2.比较
10         JG    4B             C          如果 K>KEY(P)则转到 T4
11         JE    SUCCESS        C1         如果 K = KEY(P)则转出
12         LD2   0,1(LLINK)     C1 - S     T3.左移,Q←LLINK(P)
13         J2NZ  1B             C1 - S     如果 Q≠Λ 则转到 T2
14  5H     LD2   AVAIL          1 - S      T5.插入树中
15         J2Z   OVERFLOW       1 - S
16         LDX   0,2(RLINK)     1 - S
17         STX   AVAIL          1 - S      Q⇐AVAIL
18         STA   1,2            1 - S      KEY(Q)←K
19         STZ   0,2            1 - S      LLINK(Q)←RLINK(Q)←Λ
20         JL    1F             1 - S      是 K<KEY(P)吗?
21         ST2   0,1(RLINK)     A          RLINK(P)←Q
22         JMP   * +2           A
23  1H     ST2   0,1(LLINK)     1 - S - A  LLINK(P)←Q
24  DONE   EQU   *              1 - S      插入后转出
```
▌

这个程序的前 13 行进行查找;后 11 行进行插入。查找阶段的运行时间是$(7C + C1 - 3S + 4)u$,其中

C = 所作比较数;

$C1 = K \leqslant$ KEY(P)的次数;
$C2 = K >$ KEY(P)的次数;
$S = 1$[查找是成功的]。

平均说来,我们有 $C1 = \frac{1}{2}(C+S)$,因为 $C1 + C2 = C$,且 $C1 - S$ 有和 $C2$ 相同的概率分布。所以运行时间大约是$(7.5C - 2.5S + 4)u$。这优于使用一株隐含树的二分查找算法(参见程序 6.2.1C)。通过像在程序 6.2.1F 中那样拷贝代码,我们可以消去程序 T 的行 08,使运行时间减少到$(6.5C - 2.5S + 5)u$。如果查找是不成功的,则程序的插入阶段额外耗费时间 $14u$ 或 $15u$。

算法 T 很容易修改为适用于可变长的键码和可变长的记录。例如,如果我们以后进先出方式顺序地分配可利用的空间,则不难建立大小可变的节点;(1)中的头一个字可指出大小。由于以树为基础的符号表算法能有效地使用存储,在编译程序、汇编程序以及装入程序中使用这种方法是特别有吸引力的。

但最坏情况如何呢? 当程序员们第一次看到算法 T 时,通常是怀疑它的。如果图 10 的键码是以字符顺序 AQUARIUS,…,VIRGO 而不是以历法顺序 CAPRICORN,…,SCORPIO 记入树中的,则算法将构造出一个实质上确定一个顺序查找的蜕化树,所有 LLINKs 将是空的。类似地,如果键码以古怪的次序

AQUARIUS,VIRGO,ARIES,TAURUS,CANCER,SCORPIO,CAPRICORN,PISCES,GEMINI,
LIBRA,LEO

出现,则我们得到同样坏的“Z”字形弯弯曲曲的树。(请试一下!)

另一方面,图 10 中特殊的树,对一次成功的查找,平均仅需要 $3\frac{2}{11}$次比较;这仅仅比最好的二叉树中所能达到的极小平均比较数 3 稍大一点。

当我们有了一株相当平衡的树时,查找时间大体上与 $\log N$ 成正比。但当我们有一株蜕化树时,查找时间大体却与 N 成正比。习题 2.3.4.5-5 证明,如果把每株 N 节点的二叉树都当作是同等可能的,则平均查找时间将大约同$\sqrt{N}$成正比。对于算法 T 我们实际上能预期什么特性呢?

幸而,可以证明,如果诸键码是以随机次序插入树中的,则树查找将仅需要大约 $2\ln N \approx 1.386 \lg N$ 次比较。平衡树是普遍的,而蜕化树则是非常稀少的。

关于这一事实,出人意料,有一个简单的证明。我们假定,在 N 个键码的 $N!$种可能的次序中,每一种都有同等可能用作插入序列以构造一株树。找出一个键码所需要的比较次数,恰比把此键码记入树所需要的比较次数多 1。因此,如果 C_N 是一次成功的查找中的平均比较次数,且 C'_N 是在一次不成功的查找中的平均比较次数,则我们有

$$C_N = 1 + \frac{C'_0 + C'_1 + \cdots + C'_{N-1}}{N} \tag{2}$$

但是,内部和外部路径长度之间的关系式告诉我们

$$C_N = \left(1 + \frac{1}{N}\right)C'_N - 1 \tag{3}$$

就是等式 6.2.1-(2)。把(3)同(2)合在一起得到

$$(N+1)C'_N = 2N + C'_0 + C'_1 + \cdots + C'_{N-1} \tag{4}$$

这个递推式容易解出。减少等式

$$NC'_{N-1} = 2(N-1) + C'_0 + C'_1 + \cdots + C'_{N-2}$$

得到

$$(N+1)C'_N - NC'_{N-1} = 2 + C'_{N-1}$$

因此

$$C'_N = C'_{N-1} + 2/(N+1)$$

由于 $C'_0 = 0$,这意味着

$$C'_N = 2H_{N+1} - 2 \tag{5}$$

应用式(3)并进行简化即得到所希望的结果

$$C_N = 2\left(1 + \frac{1}{N}\right)H_N - 3 \tag{6}$$

下面的习题 6、7 和 8 给出更详细的信息;有可能计算 C_N 和 C'_N 精确的概率分布,而不仅仅是平均值。

树插入排序 算法 T 是为了进行查找而设计的,但它也可用作内部排序算法的基础;事实上,我们可以把它看作表插入算法 5.2.1L 的一种自然推广。如果正确地编制程序,则它的运行时间将只比我们在第 5 章中所讨论的某些最好的算法略慢一点。在把所有的键码构造成一株树之后,一个对称的树遍历过程(算法 2.3.1T)将按排序次序访问这些记录。

然而,一些预防措施是必要的。如果在步骤 T2 中 $K = \text{KEY(P)}$,则需要完成一些不同的事情,因为我们正在排序而不是在进行查找。一种解决方法是把 $K = \text{KEY(P)}$ 当成同 $K > \text{KEY(P)}$ 完全一样,这就得到了一个稳定的排序方法。(相等的键码在树中未必是相邻的,它们只不过是在对称的次序下相邻而已)。但如果出现许多重复的键码,这个方法会使树变得极不平衡,因而排序将减慢。另一种想法是对每一个节点,都保持着具有相同键码的所有记录的一张表;这虽需要另一个链接字段,但当出现大量相同的键码时能更快地排序。

于是,如果我们仅对排序感兴趣,而对查找不感兴趣,则算法 T 不算最好但也不坏。而且,如果有把查找和排序组合起来的应用,树方法是可以热情地推荐的一种方法。

说明这一点是有趣的,即在树插入排序的分析与快速排序的分析之间有着很强的关系,尽管这些方法表面上毫无类似之处。如果我们把 N 个键码逐次地插入到开始为空的一株树中,则除了少数例外,我们所做的键码之间的平均比较次数同算法 5.2.2Q 的相同。例如,在树插入中每个键码同 K_1 进行比较,而后,小于 K_1 的每个键码同小于 K_1 的第一个键码进行比较,等等;在快速排序中,每一个键码同头一个分划元素 K 进行比较,而后,小于 K 的每个键码须同小于 K 的一个特殊元素进

行比较,等等。在这两种情况下所作的平均比较数都是 $NC_N - N$。(然而,算法 5.2.2Q 实际上稍微多做了一点比较,为的是加速内循环)。

删去 有时我们要使计算机忘记它所知道的表中的一项。我们能很容易地删去其中 LLINK = Λ 或 RLINK = Λ 的一个节点;但当两个子树都是非空时,就要做些特殊处理,因为我们不能一次指向两条路。

例如,再次考虑图 10;我们如何删去根节点摩羯座(CAPRICORN)呢? 一个解决方法是删去在字典次序下的下一个节点,它总有一个空的 LLINK,然后重新把它插入以代替我们真正要删去的节点。例如,在图 10 中,我们可以删去双子座(GEMINI),然后以双子座(GEMINI)来代替摩羯座(CAPRICORN)。这个操作保持了表中各项必要的自左到右的次序。下列算法给出了这样一个删去过程的详细描述。

算法 D (*树删去*) 设 Q 是一个变量,它指向如算法 T 中所表示的二分查找树的一个节点。本算法删去该节点,保留一株二分查找树。(实际上,在树的某个节点中,我们将有Q = ROOT 或 Q≡LLINK(P)或 Q≡RLINK(P)。这个算法在存储中重置 Q 的值,以反映删去)。

D1.[RLINK 为空吗?] 置 T←Q。如果 RLINK(T) = Λ,则置 Q←LLINK(T)并转到 D4。(例如,如果对于某个 P,Q≡RLINK(P),则我们将置 PLINK(P)←LLINK(T))。

D2.[找后继者] 置 R←RLINK(T)。如果 LLINK(R) = Λ,则置 LLINK(R)←LLINK(T),*Q*←*R*,并转到 *D*4。

D3.[找空的 LLINK] 置 S←LLINK(R)。然后,如果 LLINK(S)≠Λ,则置 R←S 并重复这一步骤直到 LLINK(S) = Λ 为止。(这时,S 将等于 Q$,即 Q 的对称的后继者)。最后,置 LLINK(S)←LLINK(T),LLINK(R)←RLINK(S),RLINK(S)←RLINK(T),Q←S。

D4.[释放此节点] 置 AVAIL⇐T,于是把删去的节点送回到可用存储空间中。 ▌

读者可能希望通过删去图 10 中的宝瓶座(AQUARIUS)、巨蟹座(CANCER)和摩羯座(CAPRICORN)来试验这个算法;每种情况是稍微不同的。一个机灵的读者可能已经注意到,对于 RLINK(T)≠Λ,LLINK(T) = Λ 的情况没有进行任何特殊的判断;我们将把这一情况的讨论推迟到稍后进行,因为这一算法有着某些非常有趣的性质。

由于算法 D 左右两边很不对称,有理由认为一连串的随机删去和插入将使树失去平衡,从而使我们已经作出的有效性估计成为不正确的。但是事实上,删去绝不会使树蜕化!

定理 H (T.N.Hibbard,1962) *通过算法 D 从一株随机树删去一个随机元素之后,得到的树仍是随机的*。

[不是学数学的读者,请跳到(10)。] 当然,要承认这个定理的叙述是非常含混的。我们可以更精确地概述这一情况如下:设 T 是 n 个元素的一株树,且 $P(T)$是

当由算法 T 以随机次序插入 T 的键码时，T 出现的概率。有些树的出现概率可能比其它大些。命 $Q(T)$是通过算法 T 以随机次序插入 $n+1$ 个元素，而后通过算法 D 随机地选择其中一个元素并把它删去时，T 将出现的概率。在计算 $P(T)$时，我们假定键码的 $n!$种排列是同等可能的；在计算 $Q(T)$时，假定原来的 n 个键码和准备删去的键码的$(n+1)!(n+1)$种排列是同等可能的。此定理指出，对所有 T，$P(T)=Q(T)$。

证明　我们面对着这样一个事实，即诸排列是同等可能的，而诸树不一定，因此我们将通过把排列当作随机对象来证明这个结果。下面，给出从排列中删去的定义，而后再证明"从一个随机排列中删去一个随机元素后仍保留一个随机排列"。

设 $a_1a_2\cdots a_{n+1}$是$\{1,2,\cdots,n+1\}$的一个排列；我们要定义删去 a_i 的操作，以便得到$\{1,2,\cdots,n\}$的一个排列 $b_1b_2\cdots b_n$。这个操作应该对应于算法 T 和 D，使得如果我们首先通过插入序列 $a_1,a_2,\cdots,a_{n+1}$构造一株树，然后删去 a_i；并对剩下的键码按 1～n 的次序重新编号时，便得到由 $b_1b_2\cdots b_n$ 构造的树。

不难定义这样一个删去操作。有下列两种情况。

情况 1：$a_i=n+1$，或对某个 $j<i$，$a_i+1=a_j$。（这实质上是条件"RLINK(a_i)$=\Lambda$"）。从这个序列撤销 a_i，并把每个大于 a_i 的元素减 1。

情况 2：对某个 $j>i$，$a_i+1=a_j$。以 a_j 代替 a_i，从 a_j 原来的位置撤销 a_j，并把每个大于 a_i 的元素减 1。

例如：假设我们有排列 4 6 1 3 5 2。如果把待删去的元素画上圆圈，则有

④ 6 1 3 5 2 = 4 5 1 3 2	4 6 1 ③ 5 2 = 3 5 1 4 2
4 ⑥ 1 3 5 2 = 4 1 3 5 2	4 6 1 3 ⑤ 2 = 4 5 1 3 2
4 6 ① 3 5 2 = 3 5 1 2 4	4 6 1 3 5 ② = 3 5 1 2 4

因为有$(n+1)!(n+1)$个可能的删去操作，所以如果能够证明$\{1,2,\cdots,n\}$的每一排列恰是$(n+1)^2$ 次删去的结果，则定理得证。

设 $b_1b_2\cdots b_n$ 是$\{1,2,\cdots,n\}$的一个排列。我们将定义$(n+1)^2$ 次删去，其中满足 $1\leqslant i,j\leqslant n+1$ 的每一对偶 i,j 对应一次删去，如下：

如果 $i<j$，则删去是

$$b'_1\cdots b'_{i-1}\,\textcircled{b_i}\,b'_{i+1}\cdots b'_{j-1}(b_i+1)b'_j\cdots b'_n \tag{7}$$

这里如以下所示，依赖于 b_k 是否小于划了圈的元素，b'_k 表示 b_k 或 b_k+1。这个删去对应于情况 2。

如果 $i>j$，则删去是

$$b'_1\cdots b'_{i-1}\,\textcircled{b_j}\,b'_i\cdots b'_n \tag{8}$$

这个删去满足情况 1 的定义。

最后,如果 $i=j$,则我们又有另一个情况 1 的删去,即

$$b'_1\cdots b'_{i-1}\ \textcircled{n+1}\ b'_i\cdots b'_n \tag{9}$$

例如,设 $n=4$ 并考虑映射为 3 1 4 2 的 25 次删去

	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$
$j=1$	⑤ 3 1 4 2	4 ③ 1 5 2	4 1 ③ 5 2	4 1 5 ③ 2	4 1 5 2 ③
$j=2$	③ 4 1 5 2	3 ⑤ 1 4 2	4 2 ① 5 3	4 2 5 ① 3	4 2 5 3 ①
$j=3$	③ 1 4 5 2	4 ① 2 5 3	3 1 ⑤ 4 2	3 1 5 ④ 2	3 1 5 2 ④
$j=4$	③ 1 5 4 2	4 ① 5 2 3	3 1 ④ 5 2	3 1 4 ⑤ 2	4 1 5 3 ②
$j=5$	③ 1 5 2 4	4 ① 5 3 2	3 1 ④ 2 5	4 1 5 ② 3	3 1 4 2 ⑤

划圈的元素总在位置 i 处,且对于固定的 i 我们已经构造了 $n+1$ 个不同的删去,每个 j 对应一个删去;因此对每个排列 $b_1b_2\cdots b_n$ 已经构造了$(n+1)^2$ 个不同的删去。因为仅仅可能有$(n+1)^2n!$个删去,故我们必然已经找到了它们的全部。▌

定理 H 的证明不仅告诉我们删去的结果,而且也有助于我们分析一次删除中的平均运行时间。习题 12 证明,当从一个随机表中删去一个随机元素时,平均说来,我们可期望执行步骤 D2 的次数稍少于这时间的一半。

现在考虑步骤 D3 中循环执行的频繁程度:假设我们正在删去 l 级上的一个节点,而且在对称次序下直接跟随的外部节点在 k 级上。例如,如果我们从图 10 删去摩羯座(CAPRICORN),则有 $l=0$ 和 $k=3$,因为节点[4]在级 3 上。如果 $k=l+1$,则在步骤 D1 中有 RLINK(T) = Λ;而如果 $k>l+1$,则我们将在步骤 D3 中置 S←LLINK(R)恰好 $k-l-2$ 次,l 的平均值是(内部路径长度)/N;k 的平均值是(外部路径长度 − 到最左外部节点的距离)/N。到最左外部节点的距离是在插入序列中自左至右极小值的个数,所以,按 1.2.10 小节的分析,该距离的平均值为 H_N。由于外部路径长度减去内部路径长度是 $2N$,故 $k-l-2$ 的平均值是 $-H_N/N$。这个值加上 $k-l-2$ 为 -1 的平均次数,我们看出,在一个随机删去中,*步骤 D3 中的操作 S←LLINK(R)平均仅被实施*

$$\frac{1}{2}+\left(\frac{1}{2}-H_N\right)\Big/N \tag{10}$$

次。这使人放心了,因为最坏的情况可能是相当慢的(见习题 11)。

尽管在我们已经叙述过的精确形式下,定理 H 是严格地正确的,但它不能如我们可能预期的那样,应用到后边跟有插入的删去序列上。在删去之后树的形状是随机的,但在一个给定树形中值的相对分布可以变化,结果是,在删去之后的头一次随机插入实际上破坏了这些形状的随机性。这个由 Gary Knott 在 1972 年首先发现的令人吃惊的事实,必须看成是可信的(参见习题 15)。甚至更令人吃惊的是由 (J. L. Eppinger[*CACM* **26** (1983),663~669 27 (1984),235]所收集到的经验证据,他发现当作了一些随机删去和插入之后,路径长度稍微减少,但在实施了大约 N^2 个删去/

插入操作之后,它却增加直到达到一个稳定的状态。当 N 大于大约 150 时,这个稳定的状态比一个随机树的特性还坏,由 Culberson 和 Munro [*Comp. J.* **32**(1989),68~75;*Algorithmica* **5** (1990) 295~311]所作的研究已经导致了这样一个似真的猜测,即在稳定状态下的平均查找时间近似地为 $\sqrt{2N/9\pi}$。然而,Eppinger 还设计了一个简单的修改,它在算法 D 与同一个算法的左右反射之间交替;他发现,这导致了一个杰出的稳定状态,其中的路径长度减少到对于随机树它的正常值的大约 88%。关于这个特性仍然缺乏理论的说明。

如上所述,尽管当 LLINK(T) = Λ 时是变得容易的情况之一,但算法 D 对此不作检验。我们可以在 D1 和 D2 之间增加一个新步骤,即

D $1\frac{1}{2}$. [LLINK 是空的吗?] 如果 LLINK = Λ,则置 Q←RLINK(T)并转到 D4。

习题 14 说明,磁带有额外步骤的算法 D,在路径长度的意义下,使树至少保持像在原来的算法 D 中那样好,而且有时结果甚至会更好。当把这个思想同 Eppinger 的对称删去策略相结合时,对于重复的随机删去/插入操作的稳定状态路径长度减少到它的只有插入时的值的大约 86%。

访问的频率 至今我们已经假定,每个键码具有同等可能作为一个查找变元。在一种更为一般的情况下,设 p_k 是要查找第 k 个插入元素的概率,其中 $p_1+\cdots+p_N=1$。则如果我们保留随机次序的假定使树的形状保持随机且等式(5)成立的话等式(2)的一个直截了当的修改表明,在一次成功的查找中平均比较次数将是

$$1+\sum_{k=1}^{N} p_k(2H_k-2) = 2\sum_{k=1}^{N} p_kH_k - 1 \tag{11}$$

例如,如果概率遵守 Zipf 定律等式 6.1-(8),并按重要性递减的次序插入键码时,则平均比较次数减少为

$$H_N - 1 + H_N^{(2)}/H_N \tag{12}$$

(见习题 18)。这大约等于按相等频率分析所预期的比较数的一半,而且它比我们使用二分查找的比较数还要少些。

图 12 示出,当 31 个最常用的英文字按频率递减的次序记入时所得到的树。使用由 H. F. Gaines 所著*Cryptanalysis* (纽约:Dover 1956),226 所作统计,对每个字亦给出其相对频率。在这株树中,每次成功查找的平均比较次数是 4.042;而利用算法 6.2.1B 或 6.2.1C 的对应的二分查找,将需要 4.393 次比较。

最优二分查找树 这些考虑使得提出,对于给定的诸频率,给出查找诸键码的一个表的最好树的问题,变得十分自然。例如,图 13 示出对于 31 个最普通的英文词的最优树;对于一个平均的成功查找,它仅要求 3.437 次比较。

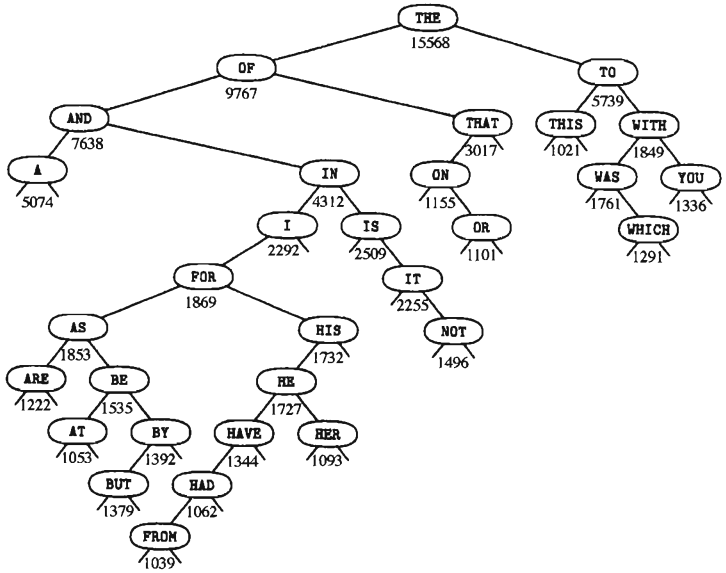


图 12　31 个最常用的英文字,以频率减少的次序插入

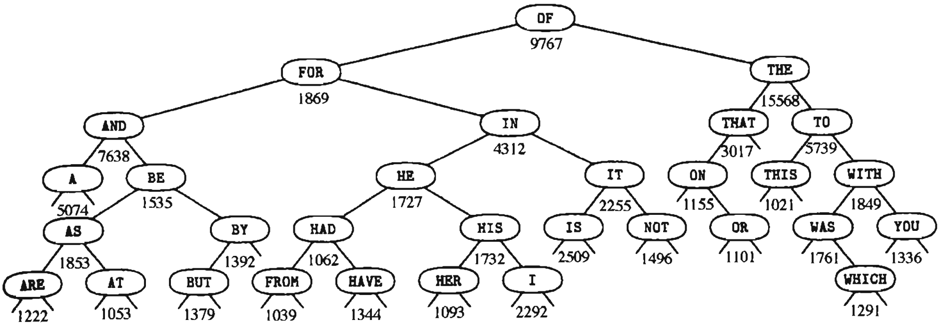
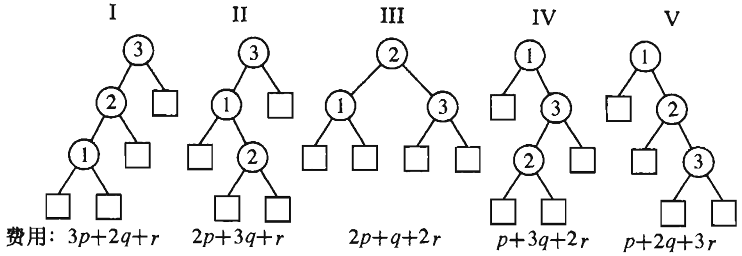


图 13　31 个最常用英文字的最优查找树

现在,我们来剖析寻道最优树的问题。例如,当 $N=3$ 时,假定键码 $K_1<K_2<K_3$ 的概率分别为 p,q,r,则有五种可能的树形:



(13)

图 14 标出了使每株树是最优的 p,q,r 的范围；如果我们随机地选择 p,q,r，则平衡的树大约有 45% 的可能是最好的(见习题 21)。

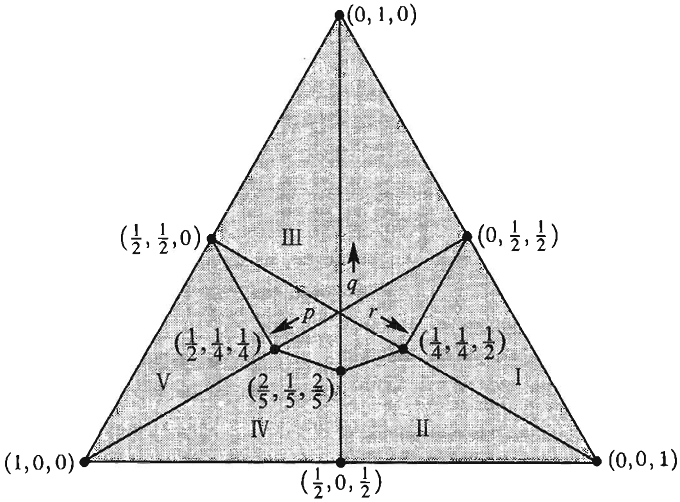


图 14　如果(K_1,K_2,K_3)的相对频率是(p,q,r)，则此图指出(13)的五株树中哪一株是最好的。虽然有三个坐标，但 $p+q+r=1$ 使这个图形变成二维的

可惜，当 N 很大时，有

$$\binom{2N}{N}\bigg/(N+1)\approx 4^N/(\sqrt{\pi}N^{3/2})$$

株二叉树，所以我们不可能全都试验它们并看出哪一株是最好的。因此我们来更仔细地研究一下最优二分查找树的性质，以便发现找出它们的一个更好的方法。

至今，我们只考虑了一次成功的查找的概率；实际上，通常也必须考虑不成功的情况。例如，图 13 中的 31 个字仅考虑了典型的英文课本中大约 36% 的情况；其余 64% 的情况肯定将影响最优查找树的结构。

因此，让我们以如下的方式来提出问题：给定 $2n+1$ 个概率 $p_1,p_2,\cdots,p_n$ 和 $q_0,q_1,\cdots,q_n$，其中

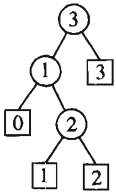
p_i = 查找变元是 K_i 的概率；

q_i = 查找变元处于 K_i 和 K_{i+1}之间的概率。

(由约定，q_0 是查找变元小于 K_1 的概率，q_n 是查找变元大于 K_n 的概率。)于是，$p_1+p_2+\cdots+p_n+q_0+q_1+\cdots+q_n=1$，我们要寻找一株二叉树，它使查找中预期的比较次数

$$\sum_{j=1}^{n}p_j(\mathrm{level}(\textcircled{j})+1)+\sum_{k=0}^{n}q_k\,\mathrm{level}(\boxed{k})\tag{14}$$

极小化，其中$\textcircled{j}$是在对称次序下的第 j 个内部节点，$\boxed{k}$是第$(k+1)$个外部节点，根的级为 0。于是，对于二叉树



(15)

预期的比较次数是 $2q_0+2p_1+3q_1+3p_2+3q_2+p_3+q_3$。我们称此为树的费用;具有极小费用的树称为是最优的。在这个定义之下,无需要求诸 p 和 q 的和为 1,对任何给定的“权”序列($p_1,\cdots,p_n;q_0,\cdots,q_n$),我们可寻求相应的极小费用的树。

我们已经在 2.3.4.5 小节中研究了构造具有极小加权路径长度树的 Huffman 过程;但该方法要求所有的 p 皆为 0,且在它产生的树上,其外部节点的权($q_0,\cdots,q_n$)通常不是严格地按从左到右的对称次序排列的。因此,我们需要另外的方法。

有一个原理能帮助我们,这就是:**一株最优树的所有子树都是最优的**。例如,如果(15)对于权($p_1,p_2,p_3;q_0,q_1,q_2,q_3$)是一株最优树,则这个根的左子树必然是对于($p_1,p_2;q_0,q_1,q_2$)为最优的;对一株子树的任何改进必然导致对整株树的改进。

这一原理提示了一个计算过程,它系统地寻道越来越大的最优子树。我们在 5.4.9 小节已经使用同样的思想来构造最优的合并形式;一般的方法称做“动态规划”,我们将在 7.7 节作进一步的考虑。

设 $c(i,j)$是具有权($p_{i+1},\cdots,p_j;q_i,\cdots,q_j$)的一株最优子树的费用;且 $w(i,j)=p_{i+1}+\cdots+p_j+q_i+\cdots+q_j$ 是所有那些权的和;于是 $c(i,j)$和 $w(i,j)$定义于 $0\leqslant i\leqslant j\leqslant n$ 上。由此得出

$$c(i,i)=0$$
$$c(i,j)=w(i,j)+\min_{i<k\leqslant j}(c(i,k-1)+c(k,j)),\quad \text{对于 } i<j \tag{16}$$

因为具有根ⓚ的一株树可能达到的极小费用是 $w(i,j)+c(i,k-1)+c(k,j)$。当 $i<j$ 时,设 $R(i,j)$是使(16)中的极小值能达到的所有 k 的集合;这个集合确定了诸最优树的可能的根。

等式(16)使我们能对 $j-i=1,2,\cdots,n$ 计算 $c(i,j)$;大约有 $\frac{1}{2}n^2$ 个这样的值,且对大约 $\frac{1}{6}n^3$ 个 k 值实现了极小化操作。这意味着我们能利用 $O(n^2)$个存储单元,在 $O(n^3)$个时间单位中确定一株最优树。

实际上,如果我们利用一个“单调性”的性质,则即可从运行时间中消去一个因子 n。设 $r(i,j)$表示 $R(i,j)$的一个元素;我们不必计算整个集合 $R(i,j)$,只计算一个代表元素就够了。一旦找到 $r(i,j-1)$和 $r(i+1,j)$,则习题 27 的结果证明,

当权非负时,我们总可以假定

$$r(i,j-1) \leqslant r(i,j) \leqslant r(i+1,j) \tag{17}$$

由于(16)中只需考察 $r(i+1,j)-r(i,j-1)+1$ 个而不是 $j-i$ 个 k 值,这就限制了极小值的查找范围。当 $j-i=d$ 时,总工作量便以缩短的级数

$$\sum_{\substack{d \leqslant j \leqslant n \\ i=j-d}} (r(i+1,j)-r(i,j-1)+1) = r(n-d+1,n)-r(0,d-1)+n-d+1 < 2n$$

为界,因此总的运行时间已减少到 $O(n^2)$。

下面的算法详细地描述了这一过程。

算法 K (*求最优的二分查找树*) 给定 $2n+1$ 个非负的权($p_1,\cdots,p_n;q_0,\cdots,q_n$),本算法构造二叉树 $t(i,j)$,它在上述定义的意义下,对于权($p_{i+1},\cdots,p_j;q_i,\cdots,q_j$)有极小的费用。要计算三个数组,即

$c[i,j]$, 对于 $0 \leqslant i \leqslant j \leqslant n$, $t(i,j)$ 的费用
$r[i,j]$, 对于 $0 \leqslant i \leqslant j \leqslant n$, $t(i,j)$ 的根
$w[i,j]$, 对于 $0 \leqslant i \leqslant j \leqslant n$, $t(i,j)$ 的总权

这个算法的结果由数组 r 指明:如果 $i=j$,则 $t(i,j)$为空;否则它的左子树是 $t(i,r[i,j]-1)$,它的右子树是 $t(r[i,j],j)$。

K1.[初始化] 对于$0\leqslant i\leqslant n$ 置$c[i,i]\leftarrow 0$,和 $w[i,i]\leftarrow q_i$,对于 $j=i+1,\cdots,n$ 置$w[i,j]\leftarrow w[i,j-1]+p_j+q_j$。然后,对于 $1\leqslant j\leqslant n$ 置$c[j-1,j]\leftarrow w[j-1,j]$以及 $r[j-1,j]\leftarrow j$(这确定所有一个节点的最优树)。

K2.[对 d 循环] 对 $d=2,3,\cdots,n$ 执行步骤 K3,然后结束该算法。

K3.[对 j 循环](我们已经确定了少于 d 个节点的最优树,这个步骤确定所有 d 个节点的最优树。)对于 $j=d,d+1,\cdots,n$ 进行步骤 K4。

K4.[求 $c[i,j]$, $r[i,j]$]置 $i\leftarrow j-d$,然后置 $c[i,j]\leftarrow w[i,j]+\min_{r[i,j-1]\leqslant k\leqslant r[i+1,j]}(c[i,k-1]+c[k,j])$。并且置 $r[i,j]$为使极小值出现的一个 k 值(习题 22 证明,$r[i,j-1]\leqslant r[i+1,j]$) ▎

作为算法 K 的一个例子,考虑图 15,它是以"上下文中的键码"(KWIC)索引应用为基础的。《*ACM* 杂志》头十卷中所有论文的标题,被排序并编制成一个重要的词汇索引,其中每个标题的每个字都有一行。然而有些字,如"THE"和"EQUATION"被认为并不包含多少信息,故从索引中省去。在图 15 的内部节点上标出了这些特殊字和它们出现的频率。注意,像"对于某个新问题的一个方程的解"这样的标题,由于完全不提供任何信息,所以根本不出现在索引中! KWIC 索引的思想,来源于 H. P. Luhn,*Amer. Documentation* **11**(1960),288~295(见 W. W. Youden,*JACM* **10** (1963),583~646,其中有完全的 KWIC 索引)。

当为排序准备一个 KWIC 索引文件时,我们可能希望用一株二分查找树,以便检验每个特定的字是否应被编入索引。其它的字落在两个未编索引的字之间,其频

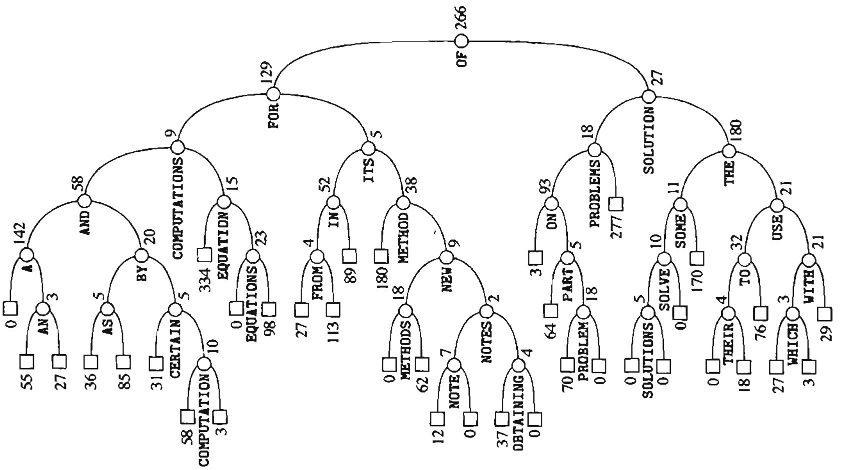


图 15 用于一个 KWIC 索引的一株最优二分查找树

率在图 15 的外部节点上标出；例如，在 1954—1963 年期间出现在 JACM 的标题中的字母处于“PROBLEMS”和“SOLUTION”之间的恰有 277 个字。

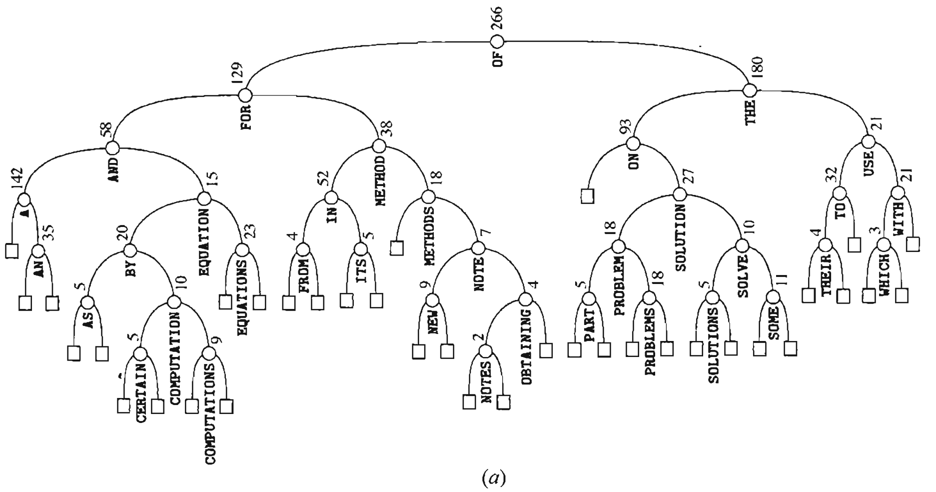
图 15 示出了 $n=35$ 时，由算法 K 得到的最优树。对于 $j=1,2,\cdots,35$，$r[0,j]$ 的计算值是(1,1,2,3,3,3,3,8,8,8,8,8,8,11,11,…,11,21,21,21,21,21,21)；对于 $i=0,1,\cdots,34$，$r[i,35]$的值是(21,21,…,21,25,25,25,25,25,25,26,26,26,30,30,30,30,30,30,30,33,33,33,35,35)。

“中间性频率”q_j 对于最优树结构产生一个值得注意的影响。图 16(a)示出了置 q_j 为 0 时所得到的最优树。类似地，内部频率 p_i 是重要的；图 16(b)示出了当 p_i 被置成 0 时的最优树。考虑全部频率的集合，平均说来，图 15 的树仅需要 4.15 个比较，而图 16 的树则分别要求 4.69 和 4.55 个比较。

由于算法 K 要求时间和空间同 n^2 成正比，因此当 n 变得很大时，它就成为不实用的了。当然，鉴于在这一章稍后有待讨论的其它查找技术，我们对于很大的 n 可能不真正使用二叉树；但仍然假定，要在 n 很大时找一棵最优的或接近于最优的二叉树。

我们已经看到，以递降的频率顺序插入键码的思想，平均说来可以产生相当好的树；但由于不利用 q_i 个权，它也可能非常之坏(见习题 20)，而且它不是经常地接近于最优。另一个方法是选择根 k，使得到的极大子树权 $\max(w(0,k-1),w(k,n))$尽可能小。这个方法也会是相当差的，因为可能选择一个具有非常小的 p_k 的节点作为根；然而，下面的定理 M 证明，得到的树将不会离最优极其遥远。

正如 W. A. Walker 和 C. C. Gotlieb[*Graph Theory and Computing*(*Academic Press*, *1972*), *303～323*]所提议的，把这两个方法组合起来，可以得到一个更令人满意的过程：尝试使左边和右边的权相等，但准备把根向左或向右移动几步，以找出具



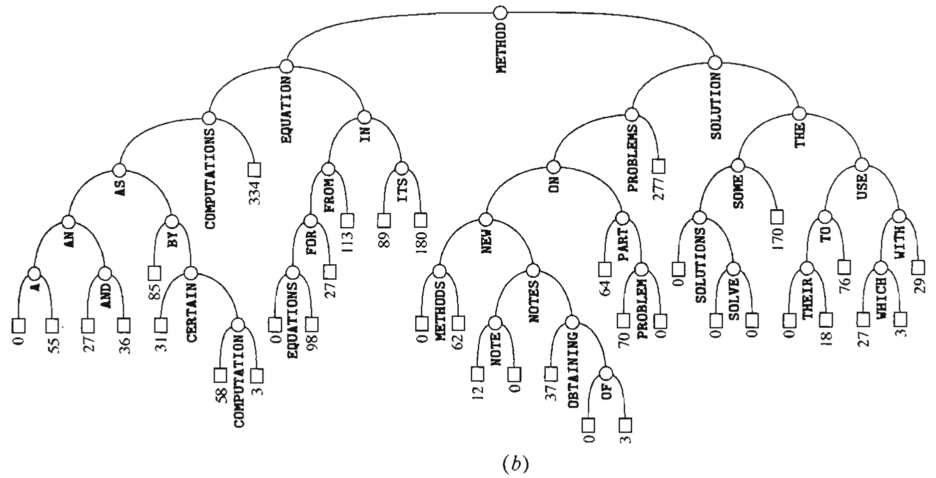


图 16 在图 15 的一半数据基础上的最优二分查找树

(a)外部频率为 0;(b)内部频率为 0。

有相当大的 p_k 的一个节点。图 17 示出为什么这个方法是有道理的:如果对图 15 的 KWIC 数据,把 $c(0,k-1)+c(k,n)$ 画作 k 的一个函数,则我们看到结果对于 p_k 的数量级是十分敏感的。

当 n 很大时,这样一个“由顶向下”的方法,可用来选择根,然后再处理左子树和右子树。当我们达到一株充分小的子树时,就可以应用算法 K。这样的方法产生相当好的树(据报告与最优的树相差不到百分之二或三),并且它只需要 $O(n)$ 个空间单位和 $O(n\log n)$ 个时间单位。事实上,M. Fredman 已经证明, 如果使用正确的数

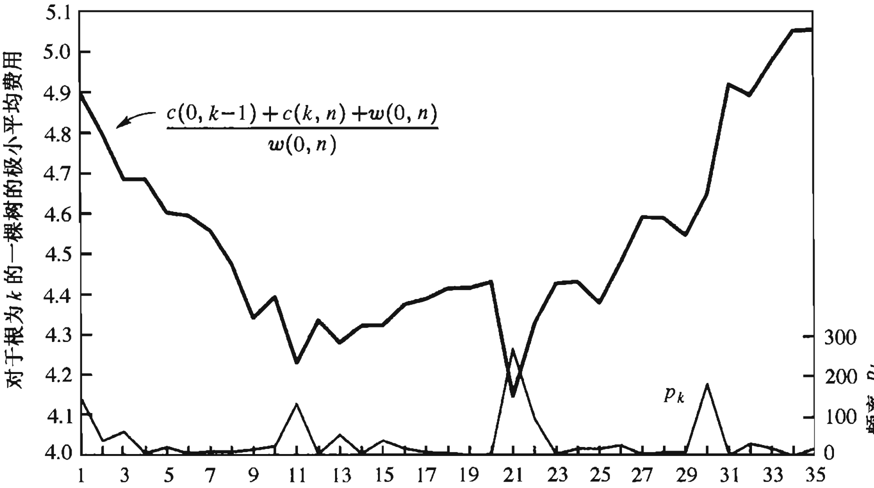


图 17　作为根 k 的一个函数的费用特性

据结构，$O(n)$个时间单位就足够[*STOC* **7**(1975)，240～244]。参见 K. Mehlhorn, *Data Structures and Algorithms* 1 (Springer，1984)，4.2 节。

最优树与熵　极小费用和称做熵的一个数学概念密切相关，它是由 Claude Shannon 在他关于信息论的杰出论文中引进的[*Bell System Tech. J.* **27** (1948)，379～423，623～656]。如果 $p_1, p_2, \cdots p_n$ 是概率且 $p_1+p_2+\cdots+p_n=1$。我们由公式

$$H(p_1,p_2,\cdots,p_n) = \sum_{k=1}^{n} p_k \lg \frac{1}{p_k} \tag{18}$$

定义熵 $H(p_1,p_2,\cdots,p_n)$。直观地说，如果 n 个事件是可能的且第 k 个事件以概率 p_k 出现，我们可以想像，当第 k 个事件已出现时，我们已接收了 $\lg(1/p_k)$位的信息。(概率为$\frac{1}{32}$的一个事件给出 5 个信息位，等等)。于是，$H(p_1,p_2,\cdots,p_n)$是在一个随机事件中预期的信息的位数。如果 $p_k=0$，我们定义 $p_k\lg(1/p_k)=0$，因为

$$\lim_{\varepsilon\to 0+} \varepsilon \lg \frac{1}{\varepsilon} = \lim_{m\to\infty} \frac{1}{m}\lg m = 0$$

这个约定允许我们在某些概率为 0 时来使用(18)。

函数 $x\lg(1/x)$是凹的；即，它的二阶导数 $-1/(x\ln 2)$为负。因此，当 $p_1=p_2=\cdots=p_n=1/n$ 时，$H(p_1,p_2,\cdots,p_n)$的极大值出现。即

$$H\left(\frac{1}{n},\frac{1}{n},\cdots,\frac{1}{n}\right) = \lg n \tag{19}$$

一般地说，如果我们确定 $p_1,p_2,\cdots,p_{n-k}$，但其它概率 $p_{n-k+1},\cdots,p_n$ 变化，我们有

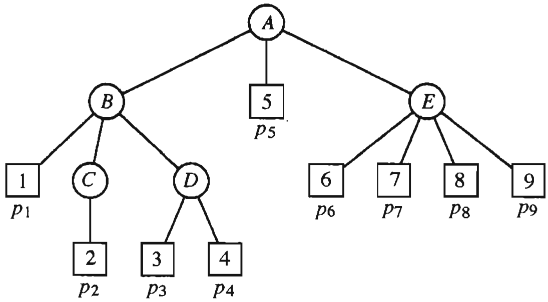
$$H(p_1,\cdots,p_{n-k},p_{n-k+1},\cdots,p_n) \leqslant H(p_1,\cdots,p_{n-k},\frac{q}{k},\cdots,\frac{q}{k}) =$$

$$H(p_1,\cdots,p_{n-k},q)+q\lg k \tag{20}$$

$$H(p_1,\cdots,p_{n-k},p_{n-k+1},\cdots,p_n)\geqslant$$
$$H(p_1,\cdots,p_{n-k,q},0,\cdots,0)=$$
$$H(p_1,\cdots,p_{n-k,q}) \tag{21}$$

其中 $q=1-(p_1+\cdots+p_{n-k})$

考虑任何无需为二叉的树,其中已对诸叶指定概率,比如说



(22)

这里 p_k 表示一个查找过程将在叶 $\boxed{k}$ 处结束的概率。于是在每个内(非叶)节点处的分支对应于在每个分支之下基于叶概率之和的一个局部概率。例如,在节点 A,以分别的概率

$$(p_1+p_2+p_3+p_4,p_5,p_6+p_7+p_8+p_9)$$

取第一个、第二个和第三个分支,而且在节点 B,概率是

$$(p_1,p_2,p_3+p_4)/(p_1+p_2+p_3+p_4)$$

我们说,每个内节点有它的局部概率分布的熵;于是

$$H(A)=(p_1+p_2+p_3+p_4)\lg\frac{1}{p_1+p_2+p_3+p_4}+$$
$$p_5\lg\frac{1}{p_5}+(p_6+p_7+p_8+p_9)\lg\frac{1}{p_6+p_7+p_8+p_9}$$

$$H(B)=\frac{p_1}{p_1+p_2+p_3+p_4}\lg\frac{p_1+p_2+p_3+p_4}{p_1}+\frac{p_2}{p_1+p_2+p_3+p_4}\lg\frac{p_1+p_2+p_3+p_4}{p_2}+$$
$$\frac{p_3+p_4}{p_1+p_2+p_3+p_4}\lg\frac{p_1+p_2+p_3+p_4}{p_3+p_4}$$

$$H(C)=\frac{p_2}{p_2}\lg\frac{p_2}{p_2}$$

$$H(D)=\frac{p_3}{p_3+p_4}\lg\frac{p_3+p_4}{p_3}+\frac{p_4}{p_3+p_4}\lg\frac{p_3+p_4}{p_4}$$

$$H(E)=\frac{p_6}{p_6+p_7+p_8+p_9}\lg\frac{p_6+p_7+p_8+p_9}{p_6}+\frac{p_7}{p_6+p_7+p_8+p_9}\lg\frac{p_6+p_7+p_8+p_9}{p_7}+$$

$$\frac{p_8}{p_6+p_7+p_8+p_9}\lg\frac{p_6+p_7+p_8+p_9}{p_8}+\frac{p_9}{p_6+p_7+p_8+p_9}\lg\frac{p_6+p_7+p_8+p_9}{p_9}$$

引理 E　*对于一个树的所有内节点 α 所取的 $p(\alpha)H(\alpha)$ 之和，其中 $p(\alpha)$ 是达到节点 α 的概率，而 $H(\alpha)$ 是 α 的熵，等于在这些叶上的概率分布的熵。*

证明　通过由底向上的归纳法，容易确立这个恒等式。例如，关于上面的公式，我们有

$$H(A)+(p_1+p_2+p_3+p_4)H(B)+p_2H(C)+(p_3+p_4)H(D)+$$
$$(p_6+p_7+p_8+p_9)H(E)=$$
$$p_1\lg\frac{1}{p_1}+p_2\lg\frac{1}{p_2}+\cdots+p_9\lg\frac{1}{p_9}$$

所有涉及 $\lg(p_1+p_2+p_3+p_4)$，$\lg(p_3+p_4)$ 和 $\lg(p_6+p_7+p_8+p_9)$ 的项都消失。▌

作为引理 E 的一个结果，我们可以使用熵来确立对于任何二叉树的费用的一个方便的下限。

定理 B　*设 $(p_1,\cdots,p_n;q_0,\cdots,q_n)$ 是如同在算法 K 中那样非负的权，并且规格化使得 $p_1+\cdots+p_n+q_0+\cdots+q_n=1$，并设 $P=p_1+\cdots+p_n$ 是一次成功查找的概率。设*

$$H=H(p_1,\cdots,p_n,q_0,\cdots,q_n)$$

是对应的概率分布的熵，且设 C 是极小费用(14)。于是，如果 $H>2P/\mathrm{e}$，我们有

$$C\geqslant H-P\lg\frac{\mathrm{e}H}{2p}\tag{23}$$

证明　取费用 C 的一个二叉树，并对它的诸叶指定概率 q_k。在每个内节点之下，还加上一个中间分支，导致有概率 p_k 的一个新叶。然后，对得到的三叉树的内节点 α 求和 $C=\sum p(\alpha)$，以及由引理 E，得到 $H=\sum p(\alpha)H(\alpha)$。

如果 α 是内节点ⓙ，熵 $H(\alpha)$ 对应于一个三路分布，其中概率之一是 $p_j/p(\alpha)$。习题 35 证明，对于所有 $x>0$，每当 $p+q+r=1$ 时，

$$H(p,q,r)\leqslant p\lg x+1+\lg\left(1+\frac{1}{2x}\right)\tag{24}$$

因此，对于所有正的 x，我们有不等式

$$H=\sum_{\alpha}p(\alpha)H(\alpha)\leqslant\sum_{j=1}^{n}p_j\lg x+\left(1+\lg\left(1+\frac{1}{2x}\right)\right)C$$

现在选择 $2x=H/P$ 就导致所求的结果，因为对于所有 $y>0$，$\lg(1+y)\leqslant y\lg\mathrm{e}$

$$C\geqslant\frac{1}{1+\lg(1+P/H)}\left(H-P\lg\frac{H}{2P}\right)=$$
$$\frac{1}{1+\lg(1+P/H)}(H+P\lg\mathrm{e})-\frac{P}{1+\lg(1+P/H)}\lg\frac{\mathrm{e}H}{2P}\geqslant$$

$$H - P\lg\frac{\mathrm{e}H}{2P} \qquad \blacksquare$$

当熵极其低时,等式(23)不必成立。但对于 $H \geqslant 2P/\mathrm{e}$ 的情况的限制不是严格的,因为 H 的值通常接近于 $\lg n$;参见习题 37。注意这个证明并不真正使用节点自左至右的阶;对于在任何阶之下有内节点概率 p_j 和外节点概率 q_k 的任何二叉查找树下限(23)成立。

即使当我们确实坚持自左至右的阶时,熵的计算也产生距离(23)不太远的一个上限。

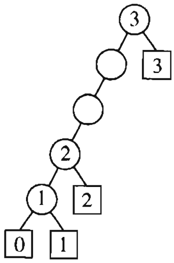
定理 M *在定理 B 的假定之下,我们也有*

$$C < H + 2 - P \tag{25}$$

证明 形成 $n+1$ 个和 $s_0 = \frac{1}{2}q_0, s_1 = q_0 + p_1 + \frac{1}{2}q_1, s_2 = q_0 + p_1 + q_1 + p_2 + \frac{1}{2}q_2, \cdots, s_n = q_0 + p_1 + \cdots + q_{n-1} + p_n + \frac{1}{2}q_n$;我们可以假定 $s_0 < s_1 < \cdots < s_n$(见习题 38)。把每个 s_k 表达为一个二进小数,如果 $s_n = 1$ 则写 $s_n = (.111\cdots)_2$。于是令串 σ_k 是 s_k 的前导二进位,对于 $j \neq k$,保留足够的二进位以区分 s_k 和 s_j。例如,我们可以有 $n=3$,而且

$$\begin{aligned} s_0 &= (.0000001)_2 & \sigma_0 &= 00000 \\ s_1 &= (.0000101)_2 & \sigma_1 &= 00001 \\ s_2 &= (.0001011)_2 & \sigma_2 &= 0001 \\ s_3 &= (.1100000)_2 & \sigma_3 &= 1 \end{aligned}$$

以如下方式构造有 $n+1$ 个叶的一个二叉树,即对于 $0 \leqslant k \leqslant n$,$\sigma_k$ 对应于从根到 $\boxed{k}$ 的通路,其中 0 表示一个左分支,而 1 表示一个右分支。还有,如果 σ_{k-1} 对于某个 α_k, β_k 和 r_k,有 $\alpha_k 0 \beta_k$ 的形式,而 σ_k 有 $\alpha_k 1 \gamma_k$ 的形式,令内节点 $\boxed{k}$ 对应于通路 α_k。于是在上面的例子中,我们将有



可能还有仍然没有名字的内节点;以它们的一个且仅一个孩子代替它们的每一个。得到的树的费用至多是 $\sum_{k=1}^{n} p_k(|\alpha_k| + 1) + \sum_{k=0}^{n} q_k |\sigma_k|$。

我们有

$$p_k \leqslant \frac{1}{2}q_{k-1} + p_k + \frac{1}{2}q_k = s_k - s_{k-1} \leqslant 2^{-|\alpha_k|} \qquad (26)$$

因为 $s_k \leqslant (.\alpha_k)_2 + 2^{-|\alpha_k|}$ 和 $s_{k-1} \geqslant (.\alpha_k)_2$。其次，如果 $q_k \geqslant 2^{-t}$；我们有 $s_k \geqslant s_{k-1} + 2^{-t-1}$，$s_{k+1} \geqslant s_k + 2^{-t-1}$，因此 $|\sigma_k| \leqslant t+1$。由此得出 $q_k < 2^{-|\sigma_k|+2}$，而且我们已经构造了费用为

$$\leqslant \sum_{k=1}^{n} p_k(1+|\alpha_k|) + \sum_{k=0}^{n} q_k|\sigma_k| \leqslant \sum_{k=1}^{n} p_k\left(1+\lg\frac{1}{p_k}\right) + \sum_{k=0}^{n} q_k\left(2+\lg\frac{1}{q_k}\right) =$$
$$P + 2(1-P) + H = H + 2 - P$$

的一株二叉树。▌

在图 15 的 KWIC 索引应用中，我们有 $P = 1304/3288 \approx 0.39659$，而且 $H(p_1, \cdots, p_{35}, q_0, \cdots, q_{35}) \approx 5.00635$。因此定理 B 告诉我们 $C \geqslant 3.3800$，而定理 M 告诉我们 $C < 6.6098$。

Garsia-Wachs 算法　在 $p_1 = \cdots = p_n = 0$ 的特殊情况下算法 K 的一个令人喜悦的改进是可能的。其中仅仅叶概率$(q_0, q_1, \cdots, q_n)$是有关的这种情况特别重要，因为它在好多重要应用中出现。因此在本小节的剩下部分我们假定，概率 p_j 全为零。注意在这种情况下定理 B 和 M 归结为不等式

$$H(q_0, q_1, \cdots, q_n) \leqslant C(q_0, q_1, \cdots, q_n) < H(q_0, q_1, \cdots, q_n) + 2 \qquad (27)$$

而且费用函数(14)简化为

$$C = \sum_{k=0}^{n} q_k l_k \qquad l_k = \boxed{k}\text{的级} \qquad (28)$$

以下发现是使一个更简单的算法成为可能的关键性质。

引理 W　*如果 $q_{k-1} > q_{k+1}$，则在每个最优树中 $l_k \leqslant l_{k+1}$，如果 $q_{k-1} = q_{k+1}$，则在某个最优树中 $l_k \leqslant l_{k+1}$。*

证明　假设 $q_{k-1} \geqslant q_{k+1}$ 并且考虑其中 $l_k > l_{k+1}$ 的一株树，于是 $\boxed{k}$ 必定是一个右儿子，而且它的左兄弟 L 是权 $c \geqslant q_{k-1}$ 的一个子树。用 L 代替 $\boxed{k}$ 的双亲；以其儿子为 $\boxed{k}$ 和 $\boxed{k+1}$ 的一个节点代替 $\boxed{k+1}$。这使整个费用改变 $-c - q_k(l_k - l_{k+1} - 1) + q_{k+1} \leqslant q_{k+1} - q_{k-1}$。所以如果 $q_{k-1} > q_{k+1}$，则给定的树不是最优的，而且如果 $q_{k-1} = q_{k+1}$，则一个最优树已被转换为另一株最优树。在后一种情况下，我们已经发现一棵最优树，其中 $l_k = l_{k+1}$。▌

对于这个结构更深的分析告诉我们更多的结果。

引理 X　*假设 j 和 k 是使得 $j < k$ 的下标。而且我们有*

i) 对于 $1 \leqslant i < k$，$q_{i-1} > q_{i+1}$，

ii) $q_{k-1} \leqslant q_{k+1}$；

iii) 对于 $j \leqslant i < k-1$，$q_i < q_{k-1} + q_k$；而且

iv) $q_{j-1} \geqslant q_{k-1} + q_k$。

于是有一株最优树，其中 $l_{k-1} = l_k$，而且两者中

a) $l_j = l_k - 1$，或者

b) $l_j = l_k$ 且$\boxed{j}$是一个左儿子。

证明：通过在引理 W 中把左和右转过来，我们看到(ii)意味着一株最优树的存在性，其中 $l_{k-1} \geqslant l_k$。引理 W 和(i)也意味着 $l_1 \leqslant l_2 \leqslant \cdots \leqslant l_k$。因此 $l_{k-1} = l_k$。

对于某个满足 $j \leqslant s < k$ 的 s，假设 $l_s < l_k - 1 \leqslant l_{s+1}$。设 t 是使得 $l_t = l_k$ 的 $<k$ 最小下标。于是 $l_{s+1} = \cdots = l_{t-1} = l_k - 1$，而且$\boxed{s+1}$是一个左儿子。因此 $t - s$ 是奇数，而且对于$i = s+1, s+3, \cdots, t$，节点$\boxed{i}$是一个左儿子。以$\boxed{t+1}$代替$\boxed{t}$的双亲；对于 $s < i < t$，以$\boxed{i+1}$代替$\boxed{i}$；而且以其子女为$\boxed{s}$和$\boxed{s+1}$的一个内节点代替外节点$\boxed{s}$。这使费用改变$\leqslant q_s - q_t - q_{t+1} \leqslant q_s - q_{k-1} - q_k$，所以如果 $q_s < q_{k-1} + q_k$，则它是一个改进。因此，由(iii)，$l_j \geqslant l_{k-1}$。

我们仍然未使用假设(iv)。如果 $l_j = l_k$，而且$\boxed{j}$不是一个左儿子，则$\boxed{j}$必定是$\boxed{j-1}$的右兄弟。以$\boxed{j-1}$代替它们的父节点，然后对于 $j < i < k$，以$\boxed{i-1}$代替叶$\boxed{i}$；而且以其子女为$\boxed{k-1}$和$\boxed{k}$的一个内节点代替外节点$\boxed{k}$。这使费用改变 $-q_{j-1} + q_{k-1} + q_k \leqslant 0$，所以我们得到满足($b$)的一株最优树。▌

引理 Y　*设 j 和 k 如同在引理 X 中那样，并考虑通过删去 q_{k-1}和 q_k，而且在 q_{i-1}之后插入 $q_{k-1} + q_k$ 得到的修改的概率* $(q'_0, \cdots, q'_{n-1}) = (q_0, \cdots, q_{j-1}, q_{k-1} + q_k, q_j, \cdots, q_{k-2}, q_{k+1}, \cdots q_n)$。于是

$$C(q'_0, \cdots, q'_{n-1}) \leqslant (q_{k-1} + q_k) + C(q_0, \cdots, q_n) \tag{29}$$

证明　只须证明，对于$(q_0, \cdots, q_n)$的任何最优树可被转换成相同费用的一株树，其中$\boxed{k-1}$和$\boxed{k}$是兄弟，而且诸叶以排列的次序

$$\boxed{0}\ \boxed{j-1}\ \boxed{k-1}\ \boxed{k}\ \boxed{j}\ \cdots\ \boxed{k-2}\ \boxed{k+1}\ \cdots\ \boxed{n} \tag{30}$$

出现。我们以在引理 X 中构造的树开始。如果它有类型(b)，我们可简单地对叶更名，把$\boxed{k-1}$和$\boxed{k}$向左滑动 $k-1-j$ 个位置，如果它是类型(a)，假设 $l_{s-1} = l_k - 1$ 和 $l_s = l_k$；我们进行如下：首先把$\boxed{k-1}$和$\boxed{k}$左滑动 $k-l-s$ 个位置；然后以$\boxed{s-1}$代替它们的(新的)双亲；最后，以其子女为$\boxed{k-l}$和$\boxed{k}$的一个节点代替$\boxed{j}$，而且对于 $j < i < s$，以$\boxed{i-1}$代替节点$\boxed{i}$。▌

引理 Z　*在引理 Y 的假设之下，(29)中的等式成立。*

证明　对于$(q'_0, \cdots, q'_{n-1})$的每株树对应于有叶(30)的一株树，其中两个越出次序的叶节点$\boxed{k-1}$和$\boxed{k}$是兄弟。令内节点 x 是它们的双亲。我们要来证明，该类

型的任何最优树可被转换成相同费用的一株树，其中叶以正常的次序$\boxed{0}\cdots\boxed{n}$出现。

如果 $j=k-1$，则无需证明任何东西。否则对于 $j\leqslant i<k-1$，我们有 $q'_{i-1}>q'_{i+1}$，因为 $q_{j-1}\geqslant q_{k-1}+q_k>q_j$。因此由引理 W，我们有 $l_x\leqslant l_j\leqslant\cdots\leqslant l_{k-2}$，其中 l_x 是 x 的级，而且对于 $j\leqslant i<k-1$，l_i 是$\boxed{i}$的级。如果 $l_x=l_{k-2}$，我们简单地把节点 x 向右滑动，以$\boxed{j}\cdots\boxed{k-2}\ x$ 来代替序列 $x\ \boxed{j}\cdots\boxed{k-2}$；这就如所求那样把叶弄好。

否则，假设 $l_s=l_x$，且 $l_{s+1}>l_x$，我们首先以$\boxed{j}\cdots\boxed{s}\ x$ 代替 $x\ \boxed{j}\cdots\boxed{s}$；这使得 $l\leqslant l_{s+1}\leqslant\cdots\leqslant l_{k-2}$，其中 $l=l_x+1$ 是节点$\boxed{k-1}$和$\boxed{k}$公共的级。最后以循环移动的序列$\boxed{s+1}\cdots\boxed{k-2}\,\boxed{k-1}\,\boxed{k}$代替节点$\boxed{k-1}\,\boxed{k}\,\boxed{s+1}\cdots\boxed{k-2}$。

习题 40 证明，这减少费用，除非 $l_{k-2}=l$。但是由于引理 Y，费用不能减少。因此 $l_{k-2}=l$，证明完成。▎

这些引理表明，对于 $n+1$ 个权 $q_0,q_1,\cdots,q_n$ 的问题可以归结为 n 个权的问题：我们首先找使得 $q_{k-1}\leqslant q_{k+1}$的最小下标 k；然后我们找使得 $q_{j-1}\geqslant q_{k-1}+q_k$ 的最大 $j<k$；然后我们从这个表中删去 q_{k-1}和 q_k，并且把和 $q_{k-1}+q_k$ 就插在 q_{j-1}之后。在 $j=0$ 或 $k=n$ 的特殊情况下，这些证明表明，我们应当就像在左和右有无穷的权 q_{-1}和 q_{n+1}存在那样进行。这些证明还表明，从新的权$(q'_0,\cdots,q'_{n-1})$得到的任何最优树 T'可以重新安排成一株树 T，它在正确的自左至右的次序下有原来的权$(q_0,\cdots,q_n)$；而且每个权在 T 和 T'两者中将出现在相同的级上。

例如，图 18 示出当诸权 q_k 是在英文正文中字符 ␣, A, B, $\cdots$, Z 的相对频率的构造。开头的一些权是

186,64,13,22,32,103,⋯

而且我们有 186>13，64>22，13<32；因此，我们以 35 代替“13,22”，在新的序列

$$186,64,35,32,103,\cdots$$

中，我们以 67 代替“35,32”并且把 67 滑动到 64 的左边，得到

$$186,67,64,103,\cdots$$

然后“67,64”变成 131，而且我们开始来考察跟随 103 的那些权。在 27 个原来的权被组合成单个权 1000 之后，逐次的组合的历史确定一个这样的二叉树，这株树的磁带权路径长度是原来问题的解。

但是图 18 中的树的叶不全处于正确的次序之下，因为当我们把 $q_{k-1}+q_k$ 向左滑动时，它们变得混乱(参见习题 41)。还有，引理 Z 的证明保证，存在一株树，它的叶处于正确的次序之下，而且恰好和混乱的权有相同的级。这个非混乱的树，即图 19，因此是最优的。由 Garsia-Wachs 算法，它是二叉树的输出。

算法 G(*Garsia-Wachs 的最优二叉树算法*)　给定一序列的非负权 $w_0,w_1,\cdots,$

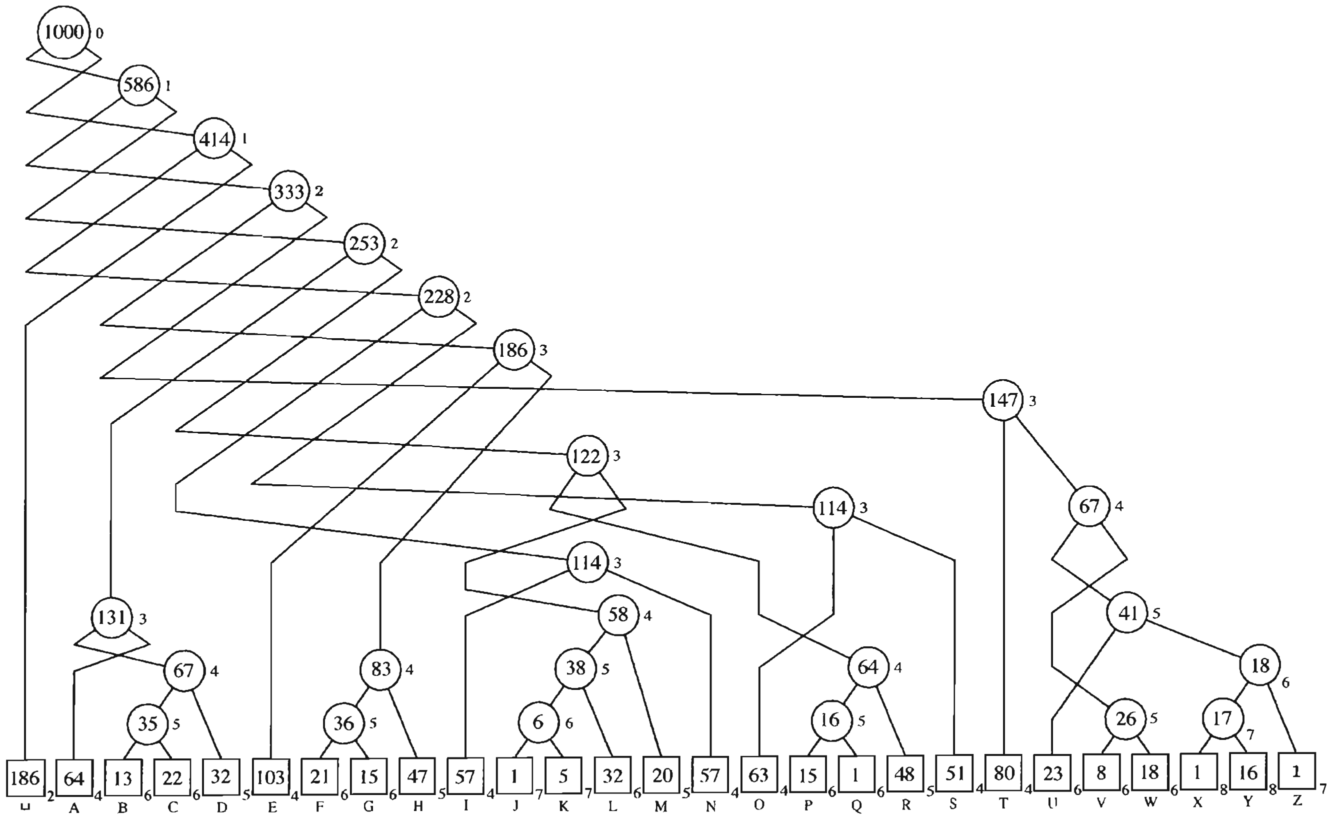


图 18　应用于字母频率数据的 Garsia－Wachs 算法:阶段 1 和 2

w_n。本算法构造出有 n 个内节点的二叉树，对于这个二叉树，$\Sigma_{k=0}^{n} w_k l_k$ 为极小。其中 l_k 是从根到外节点 $\boxed{k}$ 的距离。它使用 $2n+2$ 个节点的一个数组，对于 $0 \leqslant k \leqslant 2n+1$ 其地址是 X_k；每个节点有称做 WT，LLINK，RLINK 和 LEVEL 的四个字段。被构造的树的叶将是节点 $X_0 \cdots X_n$；内节点将是 $X_{n+1} \cdots X_{2n}$；根将是 X_{2n}；且 X_{2n+1} 被用作一个临时的标志。本算法也维持一个指针 $P_0, P_1, \cdots, P_t$ 的工作数组，其中 $t \leqslant n+1$。

G1.［开始阶段 1］　对于 $0 \leqslant k \leqslant n$，置 $\mathrm{WT}(X_k) \leftarrow w_k$ 以及 $\mathrm{LLINK}(X_k) \leftarrow \mathrm{RLINK}(X_k) \leftarrow \Lambda$。还置 $P_0 \leftarrow X_{2n+1}$，$\mathrm{WT}(P_0) \leftarrow \infty$，$P_1 \leftarrow X_0$，$t \leftarrow 1$，$m \leftarrow n$。然后对于 $j = 1, 2, \cdots, n$ 实施步骤 G2，并转到 G3。

G2.［吸收 w_j］（这时我们有基本条件

$$\mathrm{WT}(P_{i-1}) > \mathrm{WT}(P_{i+1}) \quad 对于 1 \leqslant i < t \tag{31}$$

换言之，在工作数组中的权是"2-递减的"）。实施以下的子程序 C 零次或多次，直到 $\mathrm{WT}(P_{t-1}) > w_j$。然后置 $t \leftarrow t+1$ 和 $P_t \leftarrow X_j$。

G3.［完成阶段 1］实施子程序 C 零次或多次，直到 $t = 1$。

G4.［执行阶段 2］（现在 $P_1 = X_{2n}$ 是二叉树的根，而且 $\mathrm{WT}(P_1) = w_0 + \cdots + w_n$）。对于 $0 \leqslant k \leqslant n$，置 l_k 为从节点 P_1 到节点 X_k 的距离。（参见习题 43，图 18 示出一个例子，其中级号出现在每个节点的右边）。

G5.［执行阶段 3］通过改变 $X_{n+1}, \cdots, X_{2n}$ 的链接，构造有相同级号 l_k，但有在对称次序 $X_0, \cdots, X_n$ 下的叶节点的一个新二叉树。（参见习题 44；一个例子出现于图 19 中。）▌

子程序 C（组合）本子程序是 Garsia-Wachs 算法的核心。它组合两个权，适当时把它们左移，并且维持 2－递减的条件(31)。

C1.［初始化］置 $k \leftarrow t$。

C2.［建立一个新节点］（这时我们有 $k \geqslant 2$）。置 $m \leftarrow m+1$，$\mathrm{LLINK}(X_m) \leftarrow P_{k-1}$，$\mathrm{RLINK}(X_m) \leftarrow P_k$，$\mathrm{WT}(X_m) \leftarrow \mathrm{WT}(P_{k-1}) + \mathrm{WT}(P_k)$。

C3.［向左移动跟随的节点］置 $t \leftarrow t-1$，然后对于 $k \leqslant j \leqslant t$，置 $P_{j+1} \leftarrow P_j$。

C4.［向右移前边的节点］置 $j \leftarrow k-2$；然后当 $\mathrm{WT}(P_j) < \mathrm{WT}(X_m)$ 时置 $P_{j+1} \leftarrow P_j$ 和 $j \leftarrow j-1$。

C5.［插入新节点］置 $P_{j+1} \leftarrow X_m$。

C6.［完成没有？］如果 $j > 0$ 和 $\mathrm{WT}(P_{j-1}) \leqslant \mathrm{WT}(X_m)$，则置 $k \leftarrow j$ 并且回到 C2。▌

如同前面所述，子程序 C 可能需要 $\Omega(n)$ 步来建立和插入一个新节点，因为它使用顺序的内存而不是链接表。因此算法 G 总共的运行时间可能是 $\Omega(n^2)$。但是更精心设计的数据结构可用来保证阶段 1 将至多要求 $O(n\log n)$ 步（见习题 45）。而阶段 2 和阶段 3 仅需 $O(n)$ 步。

Kleitman 和 Saks［*SIAM J. Algeb. Discr. Methods* **2**(1981)142～146］证明，最优

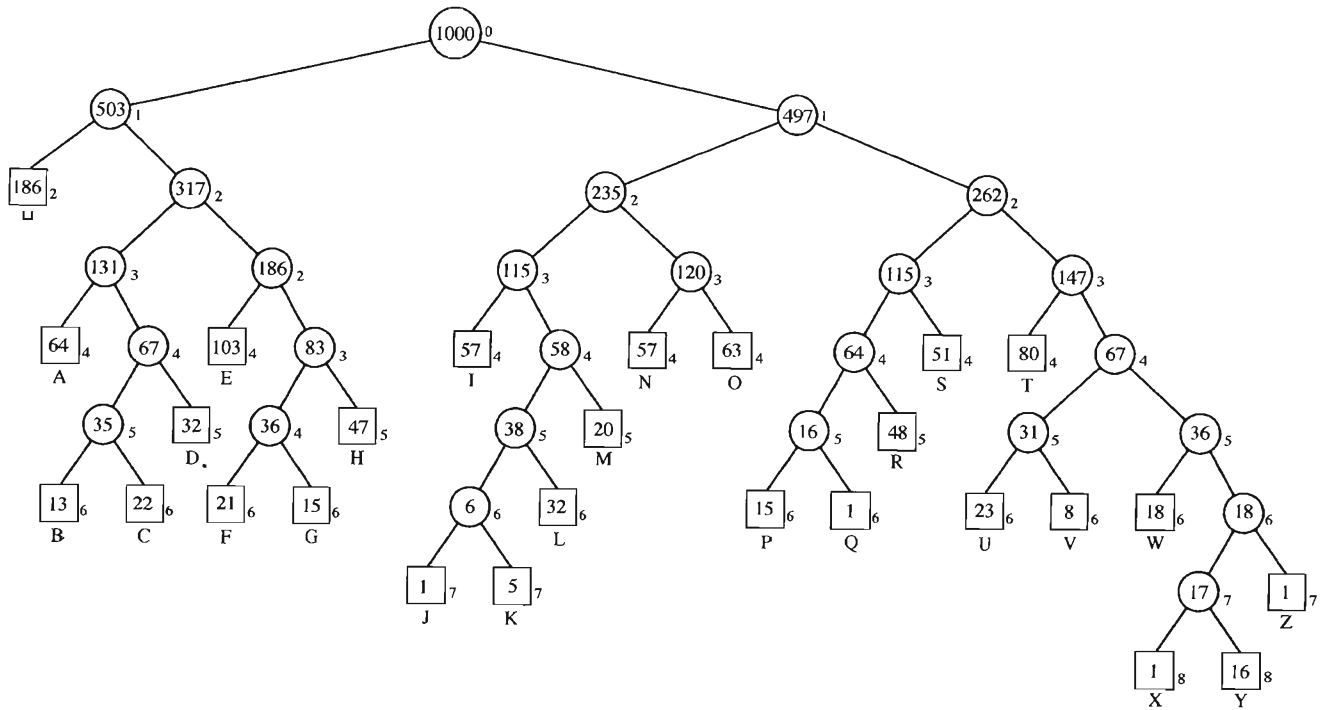


图 19　应用于字母频率数据的 Garsia－Wachs 算法：阶段 3

加权路径长度决不超过当诸 q 被安排成“锯齿次序”下出现的最优加权路径长度的值，这锯齿次序是

$$q_0 \leqslant q_2 \leqslant q_4 \leqslant \cdots \leqslant q_{2\lfloor n/2\rfloor} \leqslant q_{2\lceil n/2\rceil - 1} \leqslant \cdots \leqslant q_3 \leqslant q_1 \qquad (32)$$

(这是在习题 6.1－18 中讨论的管风琴的次序的逆)。在后一种情况下，Garsia－Wachs 算法实质上归结为在权 $q_0+q_1, q_2+q_3, \cdots$ 上的 Huffman 算法，因为在工作数组中的权实际上将是非递增的(而不仅仅像在(31)中那样是“2－递减的”)。因此我们能够改进定理 M 的上限而无须知道诸权的次序。

图 19 中的最优二叉树对编码理论以及查找有一个重要的应用。使用 0 代表树中的一个左分支，而用 1 代表右分支，我们得到下列可变长度的码字

␣	00	I	1000	R	11001
A	0100	J	1001000	S	1101
B	010100	K	1001001	T	1110
C	010101	L	100101	U	111100
D	01011	M	10011	V	111101
E	0110	N	1010	W	111110
F	011100	O	1011	X	11111100
G	011101	P	110000	Y	11111101
H	01111	Q	110001	Z	1111111

(33)

于是，像“RIGHT ON”这样一个消息将被编码为串

1100110000111010111111100010111010

尽管码字的长度是可变的，但从左至右的编码是容易的，因为树结构告知我们何时一个码字结束和另一个码字开始。这个编码方法保持了消息的字母次序，而且它平均对每个字母使用 4.2 个二进位。因此这个码可用来压缩数据文件，而不破坏字母信息的词典次序。(对于所有二叉树编码来说，每个字母 4.2 个二进位这个数字是极小的，尽管如果我们不理睬字母次序的限制，它可减少到每个字母 4.1 个二进位。如果以字母对来编码而不是以单个字母编码，可以实现进一步的减少，同时保持字母次序)。

历史和文献　这一小节的树查找法是在 20 世纪 50 年代由若干人独立地发现的。在 1952 年 8 月的未公开的备忘录中，A.I.Dumey 以下列方式描述了树插入的原始形式：

考虑其中可存储 2^n 个项目的一个磁鼓，每一个项目有一个二进地址。遵循下列程序：

1. 读入头一个项目并把它存于地址 2^{n-1} 中，即存储位置的一半处。
2. 读入下一项目，把它同头一个进行比较。
3. 如果它大，则把它放置在 $2^{n-1}+2^{n-2}$ 的位置处，如果它小，则把它放置在 2^{n-2} 处…。

树插入的另一种早期形式是由 D.J.Wheeler 提出的，他实际上允许类似于我们

将在 6.2.4 小节讨论的多路分支。一个二叉树插入技术也由 C.M.Bernes-Lee 独立地发明出来[见*Comp.J.* **2**(1959),5]。

P.E.Windley [*Comp. J.* **3** (1960), 84 ~ 88]、A.D.Booth 和 A.J.T.Colin [*Information and Control* **3** (1960), 327 ~ 334] 和 Thomas N.Hibbard [*JACM* 9 (1962),13~28]首先发表了树插入的描述。这些作者每一位似乎都已经彼此独立地发展了这个方法,而且每篇文章都以不同的方法导出了比较的平均次数(6)。各个作者还讨论了这个算法的不同方面:Windley 给出了对树插入排序的一个详细的讨论;Booth 和 Lolin 讨论了使前 2^{n-1} 个元素形成一株完全平衡的树这种预先处理的效果(见习题 4);Hibbard 提出了删去的思想并说明了树插入分析和快速排序分析之间的联系。

最优二分查找树的思想,最先是针对 $p_1 = \cdots = p_n = 0$ 的特殊情况,在研究类似(33)的字母二进编码时提出来的。E.N.Gilbert 和 E.F.Moore 的一篇非常有趣的文章[*Bell System Tech.J.* **38**(1959),933~968]中讨论了这个问题,以及它与其它编码问题的关系。Gilbert 和 Moore 证明了在特殊情况 $P=0$ 下的定理 M,并且发现利用类似于算法 K 的一个方法,但不必利用单调关系(17),就可以在 $O(n^3)$步内构造出一株最优树。K.E.Iverson[*A programming Language* (Wiley,1962),142~144]独立地考虑了其它情况,即当所有 q 都为 0 时的情况。他建议如果把根选择成使左子树和右子树的概率尽可能地相等,则将得到一株最优树;可惜,我们已经发现这个思想行不通。D.E.Knuth[*Acta Informatica* **1**(1971),14~25,270] 随后考虑了一般的 p 和 q 个权的情况,并证明了这个算法可减少到 $O(n^2)$步;他还阐述了编译程序应用中的一个例子,其中,树中的键码是在一种类似 ALGOL 语言中的"保留字"。对于 $p_j = 0$ 的情况,胡德强研究他自己的算法已经好些年了;由于这个问题的复杂性,关于这个算法正确性的一个严格证明不大好找,但是终于在 A.C.Tucker 的参与下他得到了一个证明[*SIAM J.Applied Math.* **21** (1971),514 ~ 532]。许多年以后 A.M.Garsia 和 M.L.Wachs 才发现了导致算法 G 的简化[*SICOMP* **6** (1997),622~642],尽管他们的证明仍然是比较复杂的。上面的引理 W、X、Y 和 Z 是由 J.H.Kingston 给出的[*J. Algorithms* **9** (1988), 129 ~ 136],也请参见胡德强、Kleitman 和 Tamaki 的论文,*SIAM J. Applied Math* **37** (1979),246~256,其中有胡-Tucker 算法的一个初等证明和对于其它费用函数的某些推广。

定理 B 是由 Paul J. Bayer 给出的,MIT/LCS/TM-69 报告(麻省理工学院,1975)。他还证明了定理 M 的一个稍微弱的形式,上面更强的形式是由 K.Mehlhorn 给出的[*SICOMP* **6**(1977),235~239]。

习 题

1.[*15*] 算法 T 仅仅叙述了非空树的情形。为了使它对于空树也能正确地工作,应作些什么变化?

2.[*20*] 修改算法 T 使它对右穿线树有效(参见 2.3.1 小节;在这样的树中对称遍历更容易一些)。

▶3.[*20*] 在 6.1 节中,我们发现,对顺序查找算法 6.1S 稍作修改,就能使它更快些(算法 6.1Q)。类似的技巧可以用来加速算法 T 吗?

4.[*M24*] (A.D.Booth 和 A.J.T.Colin) 给出随机顺序下的 N 个键码,假设我们使用前 2^n-1 个键码构造一株完全的平衡树,同时,对于 $0\leqslant k<n$,置 2^k 个键码于 k 级上;然后使用算法 T 来插入剩下的键码。试问在一次成功的查找中,平均比较次数是多少?[提示:修改等式(2)。]

▶5.[*M25*] 有 11! = 39916800 种不同的次序把名字 CAPRICORN、AQUARIUS 等插入到一株二分查找树中。(a)这些排列中有多少将产生图 10? (b)这些排列中有多少将产生一株蜕化的树,其中每个节点上的 LLINK 或 RLINK 为 Λ?

6.[*M26*] 设 P_{nk} 是$\{1,2,\cdots,n\}$的这样一些排列 $a_1a_2\cdots a_n$ 的数目,它们使得如果使用算法 T 逐次地把 $a_1,a_2,\cdots,a_n$ 插入到一株空树中,则当插入 a_n 时,恰好要做 k 次比较。(在这个问题中,将忽略插入 $a_1,a_2,\cdots,a_{n-1}$时所要做的比较。在正文的记号下,我们有 $C'_{n-1}=(\sum_k kP_{nk})/n!$,因为这是对一株含有 $n-1$ 个元素的树的不成功的查找中所作的平均比较次数)。

a)证明 $P_{(n+1)k}=2P_{n(k-1)}+(n-1)P_{nk}$[提示:考虑在这株树中 a_{n+1}是否落在 a_n 之下]。

b)试求生成函数 $G_n(z)=\sum_k P_{nk}z^k$ 的一个简单公式,并且借助于 Stirling 数使用你的公式来表达 P_{nk}。

c)这个分布的方差是多少?

7.[*M25*] S.R.Arora 和 W.T.Dent 在以随机顺序把 n 个元素插入到一株开始时为空的树之后,为找出第 m 个最大的元素,所需要做的平均比较次数是多少? 假定该元素的键码已给定。

8.[*M38*] 用算法 T 由 n 个随机排序的键码构造一株树,以 $p(n,k)$表示这株树的内部路径长度是 k 的概率(内部路径长度是在构造树的过程中,由树插入排序所作的比较次数)。

a)试求定义相应的生成函数的递推关系。

b)计算这个分布的方差[1.2.7 小节中的若干习题在这里可能有用!]

9.[*41*] 我们已经证明,当以随机顺序插入键码时,树的查找和插入仅需要大约 2 ln N 次比较;但实际上,顺序不能是随机的。试作一经验研究,看看树插入实际上对于一个编译程序和/或汇编程序内的符号表的适用程度如何。在典型的大型程序中的标识符,是否会产生一株相当好的平衡二分查找树?

▶10.[*22*] (R.W.Floyd) 也许我们对算法 T 的排序性质不感兴趣,但我们预料,输入将以非随机次序到来。通过使输入"看上去是"以随机次序进来的,试设计一个方法使树的查找有效。

11.[*20*] 当从大小为 N 的一株树中删去一个节点时,在步骤 D3 中实施 S←LLINK(R)的最大可能次数是多少?

12.[*M22*] 当从 N 个项目的一株随机树作一随机删去时,平均说来,步骤 D1 转到 D4 的次数是多少(见定理 H 的证明)?

▶13.[*M23*] 如果通过算法 D 删去一株随机树的根,则得到的树仍然是随机的吗?

▶14.[*22*] 证明磁带有附加步骤 D1 $\frac{1}{2}$的算法 D 产生的树的路径长度,决不多于没有这个附加步骤时产生的树的路径长度。试找出步骤 D1 $\frac{1}{2}$真正地减少路径长度的一种情况。

15.[*23*] 设 $a_1a_2a_3a_4$ 是$\{1,2,3,4\}$的一个排列,并设 $j=1,2$ 或 3。取有键码 a_1 的一个元素的树,并使用算法 T 插入 a_2,a_3;使用算法 D 删去 a_j;然后使用算法 T 插入 a_4。在 4! ×3 种可能性中有多少分别产生(13)中形状为 Ⅰ、Ⅱ、Ⅲ、Ⅳ、Ⅴ型的树?

▶16.[25] 删去操作是可交换的吗？即如果使用算法D删去 X 和 Y，则得到的树是否和使用算法D删去 Y 和 X 所得到的一样？

17.[25] 证明，如果算法D中的左和右的作用完全被颠倒，则容易推广这个算法，使得它从右穿线树删去一个给定的节点，同时保持必要的穿线(参见习题2)。

18.[M21] 证明由Zipf定律可推出(12)。

19.[M23] 当输入概率满足等式6.1-(11)中定义的"80-20"定律时，式(11)近似的平均比较次数是多少？

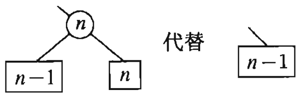
20.[M20] 假设我们以频率递减的次序 $p_1 \geqslant p_2 \geqslant \cdots \geqslant p_N$，把键码插入到一株树当中，这株树会比最优查找树坏得多吗？

21.[M20] 如果 p、q、r 是随机选择的概率，并受到条件 $p+q+r=1$ 的约束，则(13)中的树Ⅰ、Ⅱ、Ⅲ、Ⅳ和Ⅴ分别为最优的概率是多少(考虑图14中诸区域的相对面积)？

22.[M20] 证明，当实施算法K的步骤K4时，$r[i,j-1]$决不大于 $r[i+1,j]$。

▶23.[M23] 对于 $N=40$ 的情况，以及权 $p_1=9, p_2=p_3=\cdots=p_{40}=1, q_0=q_1=\cdots=q_{40}=0$，求一株最优的二分查找树(不要使用计算机)。

24.[M25] 给定 $p_n=q_n=0$，并设其它权均非负，证明 $(p_1,\cdots,p_n;q_0,\cdots,q_n)$ 的一株最优树，可以通过在 $(p_1,\cdots,p_{n-1};q_0,\cdots,q_{n-1})$ 的任何最优树中以



得到。

25.[M20] 设 A 和 B 是实数的非空集合，并且如果下列性质成立，

$(a\in A, b\in B$，且 $b<a)$蕴涵$(a\in B$，且 $b\in A)$。

则定义 $A\leqslant B$

a)证明这个关系对于非空集合是传递的。

b)证明或否定：$A\leqslant B$ 当且仅当 $A\leqslant A\cup B\leqslant B$。

26.[M22] 设 $(p_1,\cdots,p_n;q_0,\cdots,q_n)$ 是非负的权，其中 $p_n+q_n=x$。证明当 x 从0变化到 ∞，而 $(p_1,\cdots,p_{n-1};q_0,\cdots,q_{n-1})$ 保持不变时，一株最优二分查找树的费用 $c(0,n)$ 是具有整数斜率的"凹的、连续的、分段线性的"x 的函数。换言之，证明存在正整数 $l_0>l_1>\cdots>l_m$ 和实常数 $0=x_0<x_1<\cdots<x_m<x_{m+1}=\infty$，以及 $y_0<y_1<\cdots<y_m$，使得对于 $0\leqslant h\leqslant m$，当 $x_h\leqslant x\leqslant x_{h+1}$ 时，$c(0,n)=y_h+l_hx$。

27.[M33] 本题的目的是，证明每当权 $(p_1,\cdots,p_n;q_0,\cdots,q_n)$ 非负时，最优二分查找树的根 $R(i,j)$ 的集合借助于习题25中定义的关系，满足

$$R(i,j-1)\leqslant R(i,j)\leqslant R(i+1,j)\quad 对于\ j-i\geqslant 2$$

这个证明是通过对 $j-i$ 使用归纳法进行的；我们的任务是证明，假定 $n\geqslant 2$ 且对于 $j-i<n$，上面的关系成立，则 $R(0,n-1)\leqslant R(0,n)$。[由左右的对称性，得出 $R(0,n)\leqslant R(1,n)$]。

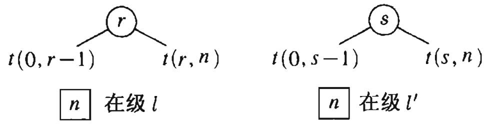
a)证明如果 $p_n=q_n=0$，则 $R(0,n-1)\leqslant R(0,n)$(见习题24)。

b)设 $p_n+q_n=x$，在习题26的记号下，设 R_h 是当 $x_h<x<x_{h+1}$ 时最优根的集合 $R(0,n)$，并设 R'_h 是当 $x=x_h$ 时最优根的集合，证明

$$R'_0\leqslant R_0\leqslant R'_1\leqslant R_1\leqslant\cdots\leqslant R'_m\leqslant R_m$$

因此，由(a)部分和习题25，对于所有的 x，我们有 $R(0,n-1)\leqslant R(0,n)$。

[提示:考虑 $x = x_h$ 的情况并假定两株树



都是最优的,且 $s < r$ 和 $l \geqslant l'$。利用归纳法假设来证明,有一株根为 ⓡ 的最优树使得 $\boxed{n}$ 在 l' 级上,以及一株根为 ⓢ 的最优树,使得 $\boxed{n}$ 在 l 级上]。

28.[*24*] 使用某种宏汇编语言来定义一个“最优二分查找”宏,其参数是一个最优二叉树的嵌套说明。

29.[*40*] 使用图 12 的频率数据,对于 31 个最常用的英文字来说,什么是最坏的二分查找树?

30.[*M34*] 证明,当 $j \geqslant i + 2$ 时,最优二分查找树的费用满足“四角不等式”

$$c(i,j) - c(i,j-1) \geqslant c(i+1,j) - c(i+1,j-1)$$

31.[*M35*](陈国财) 证明,在满足 $p_1 + \cdots + p_n + q_0 + \cdots + q_n = 1$ 的所有可能的概率集合 $(p_1,\cdots,p_n;q_0,\cdots,q_n)$ 当中,当对于所有的 i,$p_i = 0$,对于所有的偶数 j,$q_j = 0$ 以及对于所有奇数 j,$q_j = 1/\lceil n/2 \rceil$ 时,出现最昂贵的极小费用树。

▶32.[*M25*] 设 $n + 1 = 2^m + k$,其中 $0 \leqslant k \leqslant 2^m$。恰有 $\binom{2^m}{k}$ 个二叉树,其中所有外节点出现在级 m 和 $m+1$ 上。试证明,在所有这些树当中,如果我们应用算法 K 到对于充分大的 M 的权 $(p_1,\cdots,p_n;M + q_0,\cdots,M + q_n)$ 上时,我们得到对于权 $(p_1,\cdots,p_n;q_0,\cdots,q_n)$ 来说具有极小费用的一株树。

33.[*M41*] 为了找出使程序 T 的运行时间极小化的二分查找树,我们把量 $7C + C1$ 极小化,而不是简单地把比较次数 C 极小化。试给出一个算法,当这株树中的左、右分支代价不同时,它找出最优的二分查找树。[顺便指出,当右代价是左代价的两倍且节点频率都相等时,斐波那契树是最优的,参考 L.E.Stanfel,*JACM* **17**(1970),508～517。在不能一次作三路比较的机器上,算法 T 的一个程序将要在步骤 T2 中作两次比较,一次是关于等于的比较,一次是关于小于的比较;B.Sheil 和 V.R.Pratt 已经发现,这些比较不一定涉及同一个键码,所以,看来最好有一株这样的二叉树,其内节点确定一个相等测试或一个小于测试,但不是两者兼备。这种情况作为上面所述的问题的另一种可能性,对它进行探讨是很有意思的。]

34.[*HM21*] 证明当 $N \to \infty$ 时,多项式系数

$$\binom{N}{p_1 N, p_2 N, \cdots, p_n N},$$

的近似值同熵 $H(p_1,p_2,\cdots,p_n)$ 有关。

35.[*HM22*] 通过确立不等式(24)来完成定理 B 的证明。

▶36.[*HM25*](Claude Shannon) 设 X 和 Y 是具有有限范围 $\{x_1,\cdots,x_m\}$ 和 $\{y_1,\cdots,y_n\}$ 的随机变量。且设 $p_i = P_r(X = x_i)$,$q_j = P_r(Y = y_j)$,$r_{ij} = P_r(X = x_i$ 和 $Y = y_j)$。设 $H(X) = H(p_1,\cdots,p_m)$ 和 $H(Y) = H(q_1,\cdots,q_n)$ 是单个地变量分别的熵,而 $H(X,Y) = H(r_{11},\cdots,r_{mn})$ 是它们联合分布的熵。证明

$$H(X) \leqslant H(XY) \leqslant H(X) + H(Y)$$

[提示:如果 f 是任何凹函数,我们就有 $Ef(x) \leqslant f(Ex)$]。

37.[*HM26*](P.J.Bayer,1975) 假设 $(P_1,\cdots,P_n)$ 是一个随机概率分布,即对于 $1 \leqslant k \leqslant n$ 和

$P_1+\cdots+P_n=1$ 由 $p_k\geqslant 0$ 定义的在 $n-1$ 维单纯形中的一个随机点。(等价地说,$(P_1,\cdots P_n)$是在习题 3.3.2-26 意义下的随机间隔的集合。)什么是熵 $H(P_1,\cdots,P_n)$预期的值?

38.[*M20*] 尽管我们仅在 $s_0<s_1<s_2<\cdots<s_n$ 的情况下证明了定理 M,但说明为什么一般情况下它都成立?

▶39.[*M25*] 设 $w_1,\cdots,w_n$ 是满足 $w_1+\cdots+w_n=1$ 的非负的权。试证明,在 2.3.4.5 小节中所构造的 Huffman 树的加权路径长度小于 $H(w_1,\cdots,w_n)+1$。提示:参见定理 M 的证明。

40.[*M26*] 完成引理 Z 的证明。

41.[*21*] 图 18 示出了一个混乱的二叉树的构造。试在自左至右次序下列出它的叶。

42.[*23*] 说明为什么子程序 C 维持 2-递减条件(31)。

43.[*20*] 说明如何有效地实现 Garsia-Wachs 的算法阶段 2。

▶44.[*25*] 说明如何有效地实现 Garsia-Wachs 的算法阶段 3。

▶45.[*30*] 说明如何实现子程序 C 使得 Garsia-Wachs 的算法总共的运行时间至多是 $O(n\log n)$。

46.[*M30*](黄泽权和张系国) 考虑一个方案,这个方案也是通过算法 T 构造出二分查找树的,惟一区别是,当节点数达到形如 2^n-1 的一个数时,这株树被重新组织为一株完全平衡的均匀树,且对于 $0\leqslant k<n$,有 2^k 个节点在级 k 上。证明,在构造这样的树时所作的比较次数平均为 $N\lg N+O(N)$(不难证明,重新组织所需要的时间量为 $O(N)$)。

47.[*M40*] 把定理 B 和 M 从二叉树推广到 t 叉树。如果可能,也像在习题 33 中那样,允许分支费用是非均匀的。

48.[*M47*] 在随机插入和删去的条件下对于二分查找树的稳定状态进行严格的分析。

49.[*HM42*] 试分析一个随机二分查找树的平均高度。

6.2.3 平衡的树

我们刚刚学习了树插入算法,当输入数据是随机的时,这个算法将产生好的查找树,但是仍然有出现一株令人讨厌的蜕化树的可能性。也许我们可以想出一个算法,它使树总是最优的;但不幸的是,似乎这非常困难。另一个思想是,记住总的路径长度,而且当它的路径长度,比如说,超过 $5N\lg N$ 时就完全重新组织此树。但在构造这株树的过程中,这个方法可能要求大约$\sqrt{N/2}$次重新组织。

1962 年两名俄国数学家 G.M.Adelson-Velsky 和 E.M.Landis,对维持一株好查找树的问题,发现了一个非常漂亮的解[*Doklady Akademiia Nauk SSSR* **146** (1962),263~266;英文翻译在*Soviet Math*. **3**,1259~1263 上]。他们的方法仅仅要求每个节点增加额外两个二进位,并且决不使用多于O(logN)个操作来查找树或插入一个项目。事实上,我们将看到,他们的方法也导致了一项好的表示任意长度 N 的线性表的一般技术,使得下列每个操作仅以 $O(\log N)$时间单位就可完成:

i) 找出具有一给定键码的一个项目。

ii) 给定 k,找出第 k 个项目。

iii) 把一个项目插入到一个确定的位置。

iv) 删去一个确定的项。

如果我们对线性表使用顺序分配,则操作(i)和(ii)是高效的,但操作(iii)和(iv)却花费阶为 N 的步骤;另一方面,如果我们使用链接分配,则操作(iii)和(iv)是高效

的，但是操作(i)和(ii)花费阶为 N 的步骤。线性表的一种树表示可以以 $O(\log N)$ 步骤来做所有四个操作。并且它也可能相当高效地来做其它标准操作，例如，我们可以在 $O(\log(M+N))$ 步内把一个 M 个元素的表同一个 N 个元素的表链接起来。

实现所有这些的方法，涉及到我们称之为平衡树的概念。(许多作者也把它们称作 AVL 树，其中的 AV 代表 Adelson - Velsky 而 L 代表 Landis。)上一段是给平衡树做广告，它把平衡树宣扬成好似包治百病的万灵药，而使所有其它数据表示形式都相形见绌；当然，我们对于平衡树也应当有一种平衡的态度！在不涉及所有上面四个操作的应用中，我们可能达到少得多的开销和更简单的程序设计。而且，除非 N 相当大，否则平衡树并没有优点；因此，如果有花费 $64 \lg N$ 时间单位的一个高效方法，和花费 $2N$ 时间单位的一个低效方法，则我们将使用低效方法，除非 N 是大于 256 的。另一方面，N 也不应该太大；平衡树主要适合于数据的内部存储，而对于 6.2.4 小节中的外部直接存取文件，我们将研究更好的方法。由于内存储器随着时间的推移似乎变得越来越大，因此平衡树也就越来越重要。

一株树的高度定义为它最大的级，即从根到一个外节点的最长路径的长度。一株二叉树称为平衡的，如果每个节点左子树的高度同它的右子树的高度之差不超过 ±1。图 20 示出了具有 17 个内节点和高度为 5 的一株平衡树；每个节点的平衡因子用 +，· 或 - 表示，根据右子树的高度减左子树的高度是 +1、0 或 -1 而定。图 8 (6.2.1 小节)中的斐波那契树是另一株高度为 5 的平衡二叉树，它仅有 12 个内节点；在该树中的平衡因子大多是 -1。图 10(6.2.2 小节)中的黄道天宫树不是平衡的，因为在 AQUARIUS 和 GEMINI 两个节点处都不符合关于子树的高度限制。

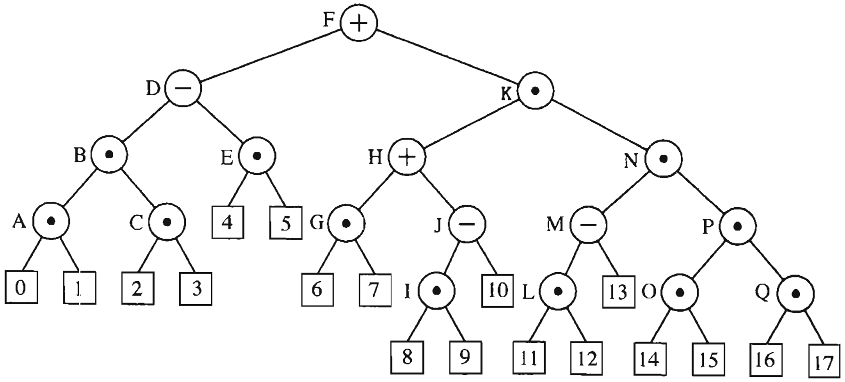


图 20　一株平衡的二叉树

平衡性这一定义表示了最优二叉树(要求所有外节点都在两个相邻的极上)和任意二叉树(没有限制)之间的折衷。因此自然要问，一株平衡树比最优树差多少。答案是，它的查找路径长度决不会比最优树长 45% 以上。

定理 A(Adelson - Velsky 和 Landis)　*具有 N 个内节点的一株平衡树的高度总*

是处于$\lg(N+1)$*和* $1.4405\lg(N+2)-0.3277$ *之间*。

证明　高度为 h 的一株二叉树显然不能有多于 2^h 个外节点;所以 $N+1\leqslant 2^h$,即在任何二叉树中 $h\geqslant\lceil\lg(N+1)\rceil$。

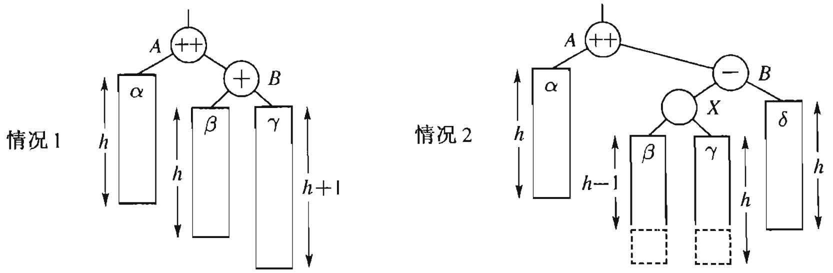
为了求 h 的极大值,我们把这个问题换一个提法,问在高度为 h 的一株平衡树中可能的极小节点数是多少。设 T_h 是具有最小节点的这样一株树;则这个根的子树之一,比如说左子树,高度为 $h-1$,而另一株子树高度为 $h-1$ 或 $h-2$。由于我们要求 T_h 的节点数为极小,所以可以假定这个根的左子树是 T_{h-1},而右子树是 T_{h-2}。这段论证表明,阶为 $h+1$ 的斐波那契树,在高度为 h 的所有可能的平衡树当中,节点数最小(见 6.2.1 小节中斐波那契树的定义)。于是

$$N\geqslant F_{h+2}-1>\phi^{h+2}/\sqrt{5}-2$$

如同在定理 4.5.3F 的推论中那样,所述结果得证。▎

这个定理的证明表明,在一株平衡树的查找中,仅当这株树至少包含 $F_{28}-1=317,810$个节点时,才需要 25 次以上的比较。

现在考虑当利用树插入(算法 6.2.2T)把一个新节点插入到一株平衡树中时,将发生什么情况。在图 20 中,如果这个新节点居于[4]、[5]、[6]、[7]、[10]或[13]处,则这株树将是平衡的。但如果新节点落在别处,则需要作某些调整。问题发生在当我们有平衡因子为 $+1$ 的一个节点时,这个节点的右子树在插入之后就更高了。或者,对偶地,如果平衡因子是 -1,则左子树就更高了。不难看到,实际上只有如下两种情况会引起麻烦:

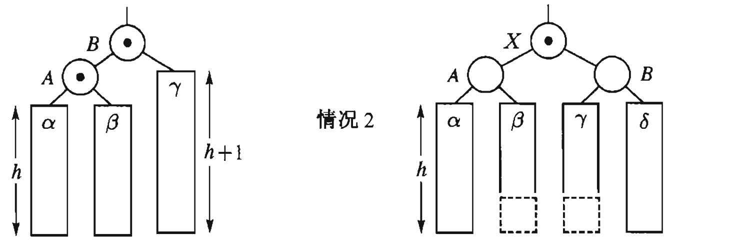


(1)

(如果我们反演这些图式,左右交换,则会出现另外两种实际上相同的情况。)在这些图式中,大的矩形 α、β、γ、δ 分别表示有图中所示高度的子树。当一个新元素刚把节点 B 的右子树的高度从 h 增加到 $h+1$ 时出现情况 1,当新元素增加 B 的左子树高度时出现情况 2。在第二种情况下,我们或者有 $h=0$(所以 X 本身是新节点),或者节点 X 有高度分别为$(h-1,h)$或$(h,h-1)$的两株子树。

在上述两种情况下,简单的变换将恢复平衡,同时保持树节点的对称次序,见(2)。

在情况 1 中,我们简单地向左"转动"树,把 β 加到 A 而不是加到 B,这个变换好像是对一个代数公式应用结合律似的,以$(\alpha\beta)\gamma$ 来代替 $\alpha(\beta\gamma)$。在情况 2 中,我们使用双重转动,首先向右转动(X,B), 然后向左转动(A,X)。在两种情况下,都只



情况 1　　情况 2　　(2)

需要改变树的少量链接。而且,新树的高度是 $h+2$,这恰是在插入之前的高度;因此原先在节点 A 上面的树的剩余部分(如果有的话),总是保持平衡的。

例如,如果我们把一个新节点插入到图 20 的位置 17 处,则在一个转动之后(情况 1)得到图 21 所示的平衡树。注意,若干平衡因子已经改变。

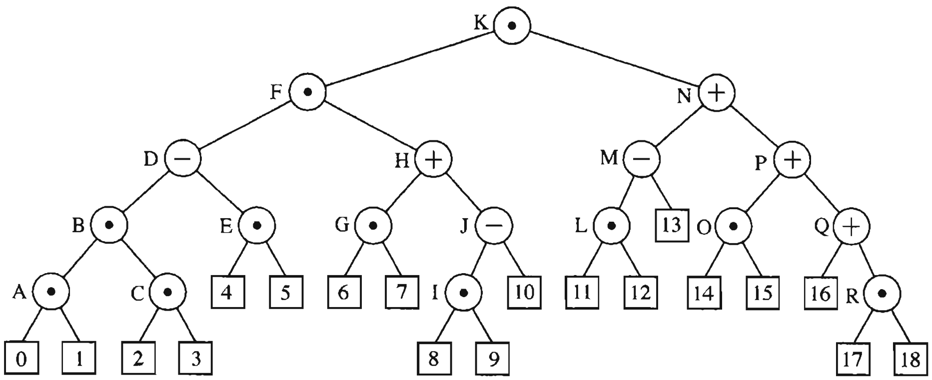


图 21　图 20 的树,在插入了新的键码 R 之后已重新平衡过

这个插入步骤的细节可以以若干方式给出。乍一看去,为了记住哪一些节点将受影响,似乎需要一个辅助的栈,但下面的算法利用了这样一个事实即在插入之前(1)中节点 B 的平衡因子为 0,从而赢得了速度。

算法 A(平衡树的查找和插入)　给定一个记录表,它形成如上所述的一株平衡的二叉树,本算法查找一个给定的变元 K。如果 K 不在表中,则把一个包含 K 的新节点插入到树的适当位置,必要时重新平衡这株树。

如同在算法 6.2.2T 中那样,假定树的节点含有 KEY、LLINK 和 RLINK 诸字段,我们还有一个新的字段。

B(P) = NODE(P) 的平衡因子

即右子树的高度减去左子树的高度;这个字段的内容总是 +1、0 或者 −1。一个特殊的表头节点也出现在树的顶上,在单元 HEAD 中;RLINK(HEAD)的值是指向树根的指针,而 LLINK(HEAD)用来记住树的整个高度。(这个算法实际上不需要知道高度,但在下面讨论的连接步骤中它是有用的)。我们假定这株树是非空的,即 RLINK

(HEAD)≠Λ。

为了叙述方便,本算法使用记号 LINK(a,P)作为 $a=-1$ 时 LLINK(P),以及 $a=+1$ 时 RLINK(P)的同义语。

A1. [初始化] 置 T←HEAD,S←P←RLINK(HEAD)。(指针变量 P 将沿树下移,S 将指向可能需要重新平衡的位置,且 T 总是指向 S 的父亲)。

A2. [比较] 如果 $K<$KEY(P),则转到 A3;如果 $K>$KEY(P),则转到 A4;如果 $K=$KEY(P),则查找成功地结束。

A3. [左移] 置 Q←LLINK(P)。如果 Q=Λ,则置 Q⇐AVAIL 和 LLINK(P)←Q 并转到步骤 A5。否则,如果 B(Q)≠0,则置 T←P 和 S←Q。最后,置 P←Q 并返回到步骤 A2。

A4. [右移] 置 Q←RLINK(P)。如果 Q=Λ,则置 Q⇐AVAIL 和 RLINK(P)←Q 并转到步骤 A5。否则,如果 B(Q)≠0,则置 T←P 和 S←Q。最后,置 P←Q 并返回到步骤 A2。(本步骤的最后部分可以同步骤 A3 的最后部分组合在一起)。

A5. [插入] (我们刚刚把一个新节点 NODE(Q)链接到这株树中,而且它的字段需要初始化。)置 KEY(Q)←K,LLINK(Q)←RLINK(Q)←Λ,B(Q)←0。

A6. [调整平衡因子] (现在在 S 和 Q 之间的诸节点上的平衡因子需要从 0 变成为 ±1。)如果 $K<$KEY(S),则置 $a\leftarrow-1$,否则置 $a\leftarrow+1$。然后置 R←P←LINK(a,S),并重复地做下列操作 0 次或多次,直到 P=Q 为止:如果 $K<$KEY(P),则置 B(P)←−1 和 P←LLINK(P);如果 $K>$KEY(P),则置 B(P)←+1 和 P←RLINK(P)。(如果 $K=$KEY(P),则置 P=Q 并且进到下一步。)

A7. [平衡动作] 现在出现若干情况:

i)如果 B(S)=0 (这株树已经长得更高),则置 B(S)←a,LLINK(HEAD)←LLINK(HEAD)+1,并结束此算法。

ii)如果 B(S)=$-a$(这株树更平衡了),则置 B(S)←0 并结束此算法。

iii)如果 B(S)=a(这株树失去了平衡),则如果 B(R)=a 转到步骤 A8,如果 B(R)=$-a$,转到步骤 A9。

(情况(iii)对应于当 $a=+1$ 时(1)中叙述的情况;S 和 R 分别指向节点 A 和 B,且 LINK($-a$,S)指向 α,等等。)

A8. [单转动] 置 P←R,LINK(a,S)←LINK($-a$,R),LINK($-a$,R)←S,B(S)←B(R)←0。转到 A10。

A9. [双转动] 置 P←LINK($-a$,R),LINK($-a$,R)←LINK(a,P),LINK(a,P)←R,LINK(a,S)←LINK($-a$,P),LINK($-a$,P)→S。现在置

$$(\mathrm{B(S)},\mathrm{B(R)})\leftarrow\begin{cases}(-a,0) & \text{如果 B(P)} = a\\(0,0) & \text{如果 B(P)} = 0\\(0,a) & \text{如果 B(P)} = -a\end{cases}\qquad(3)$$

然后置 B(P)←0。

A10. [最后一步] (我们已经完成了重新平衡的变换,把(1)变成(2),并且 P 指向新的根且 T 指向旧的子树根 S 的父亲。)如果 S=RLINK(T)则置 RLINK

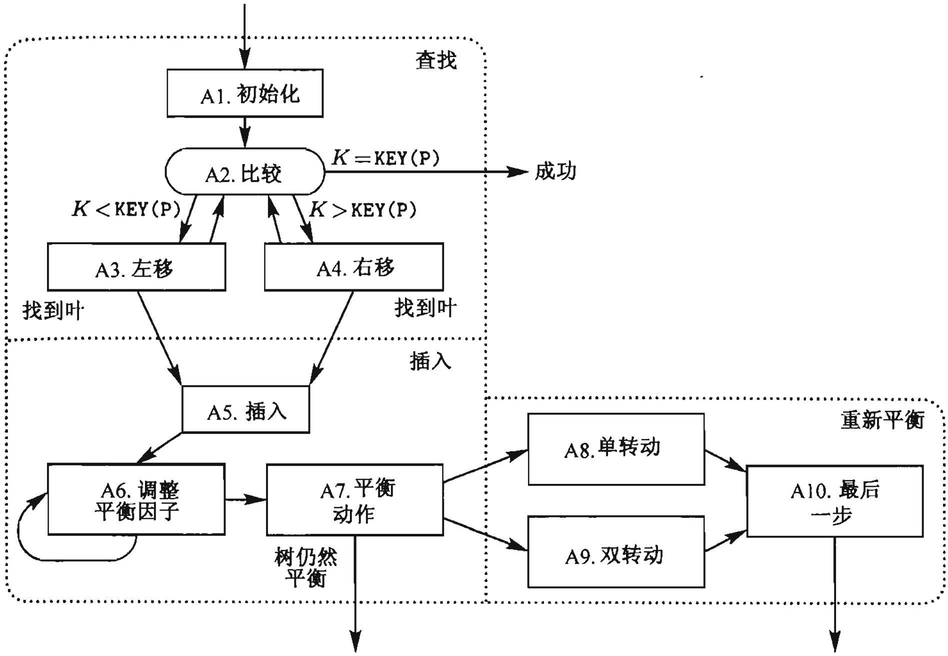


图 22　平衡树的查找与插入

(T)←P，否则置 LLINK(T)←P。 ▍

这个算法是相当长的，但它分为三个简单的部分：步骤 A1～A4 进行查找，步骤 A5～A7 插入一个新节点，而必要时步骤 A8～A10 重新平衡这株树。实际上，如果这株树是穿线的(参见习题 6.2.2-2)，则可使用同样的方法，因为平衡动作决不需要对穿线链接作困难的变动。

我们知道，这个算法花费大约 $C \log N$ 个时间单位(C 是某个常数)，但重要的是要知道 C 的近似值，使得我们能了解 N 应当多大，才值得使用平衡树。下列的 MIX 实现能增进我们对这个问题的理解。

程序 A(平衡树的查找和插入)　这个实现算法 A 的程序使用形如

B	LLINK	RLINK
KEY		

；　(4)

的树节点，rA≡K，rI1≡P，rI2≡Q，rI3≡R，rI4≡S，rI5≡T。重复步骤 A7～A9 的代码，使得 a 的值隐式地(不是显式地)出现在程序中。

```
B       EQU     0:1
LLINK   EQU     2:3
```

```
RLINK  EQU   4:5
START  LDA   K             1           A1.初始化
       ENT5  HEAD          1           T←HEAD
       LD2   0,5(RLINK)    1           Q←RLINK(HEAD)
       JMP   2F            1           转 A2 且 S←P←Q
4H     LD2   0,1(RLINK)    C2          A4.右移。Q←RLINK(P)
       J2Z   5F            C2          如果 Q=Λ 则转 A5
1H     LDX   0,2(B)        C-1         rX←B(Q)
       JXZ   *+3           C-1         如果 B(Q)=0 则转移
       ENT5  0,1           D-1         T→P
2H     ENT4  0,2           D           S←Q
       ENT1  0,2           C           P←Q
       CMPA  1,1           C           A2.比较
       JG    4B            C           如果 K>KEY(P)则转到 A4
       JE    SUCCESS       C1          如果 K=KEY(P)则转出
       LD2   0,1(LLINK)    C1-S        A3.左移。Q←LLINK(P)
       J2NZ  1B            C1-S        如果 Q≠Λ 则转移
5H     LD2   AVAIL         1-S         A5.插入
       J2Z   OVERFLOW      1-S
       LDX   0,2(RLINK)    1-S
       STX   AVAIL         1-S         Q⇐AVAIL
       STA   1,2           1-S         KEY(Q)←K
       STZ   0,2           1-S         LLINK(Q)←RLINK(Q)←Λ
       JL    1F            1-S         K<KEY(P)吗?
       ST2   0,1(RLINK)    A           RLINK(P)←Q
       JMP   *+2           A
1H     ST2   0,1(LLINK)    1-S-A       LLINK(P)←Q
6H     CMPA  1,4           1-S         A6.调整平衡因子
       JL    *+3           1-S         如果 K<KEY(S)则转移
       LD3   0,4(RLINK)    E           R←RLINK(S)
       JMP   *+2           E
       LD3   0,4(LLINK)    1-S-E       R←LLINK(S)
       ENT1  0,3           1-S         R←R
       ENTX  -1            1-S         rX←-1
       JMP   1F            1-S         转到比较循环
4H     JE    7F            F2+1-S      如果 K=KEY(P)则转到 A7
       STX   0,1(1:1)      F2          B(P)←+1(它原是+0)
```

	LD1	0,1(RLINK)	$F2$	P←RLINK(P)
1H	CMPA	1,1	$F+1-S$	
	JGE	4B	$F+1-S$	如果 $K \geqslant$ KEY(P)则转移
	STX	0,1(B)	$F1$	B(P)←−1
	LD1	0,1(LLINK)	$F1$	P←LLINK(P)
	JMP	1B	$F1$	转到比较循环
7H	LD2	0,4(B)	$1-S$	*A7. 平衡动作*。rI2←B(S)
	STZ	0,4(B)	$1-S$	B(S)←0
	CMPA	1,4	$1-S$	
	JG	A7R	$1-S$	如果 $K \geqslant$ KEY(S)则转到 $a+1$ 子程序
A7L	J2P	DONE	$U1$	如果 rI2 = $-a$ 则转出
	J2Z	7F	$G1+J1$	如果 B(S)原来是 0,则转移
	ENT1	0,3	$G1$	P←R
	LD2	0,3(B)	$G1$	rI2←B(R)
	J2N	A8L	$G1$	如果 rI2 = a 则转 A8
A9L	LD1	0,3(RLINK)	$H1$	*A9. 双转动*
	LDX	0,1(LLINK)	$H1$	LINK(a,P←LINK($-a$,R))
	STX	0,3(RLINK)	$H1$	→LINK($-a$,R)
	ST3	0,1(LLINK)	$H1$	LINK(a,P)←R
	LD2	0,1(B)	$H1$	rI2←B(P)
	LDX	T1,2	$H1$	$-a$,0 或 0
	STX	0,4(B)	$H1$	→B(S)
	LDX	T2,2	$H1$	0,0,或 a
	STX	0,3(B)	$H1$	→B(R)
A8L	LDX	0,1(RLINK)	$G1$	*A8. 单转动*
	STX	0,4(LLINK)	$G1$	LINK(a,S)←LINK($-a$,P)
	ST4	0,1(RLINK)	$G1$	LINK($-a$,P)←S
	JMP	8F	$G1$	同其它分支合并
A7R	J2N	DONE	$U2$	如果 rI2 = $-a$ 则转出
	J2Z	6F	$G2+J2$	如果 B(S)原来为 0 则转移
	ENT1	0,3	$G2$	P←R
	LD2	0,3(B)	$G2$	rI2←B(R)
	J2P	A8R	$G2$	如果 rI2 = a 则转 A8
A9R	LD1	0,3(LLINK)	$H2$	*A9. 双转动*
	LDX	0,1(RLINK)	$H2$	LINK(a,P←LINK($-a$,R))
	STX	0,3(LLINK)	$H2$	→LINK($-a$,R)

```
           ST3    0,1(RLINK)    H2      LINK(a,P)←R
           LD2    0,1(B)        H2      rI2←B(P)
           LDX    T2,2          H2      -a,0 或 0
           STX    0,4(B)        H2        →B(S)
           LDX    T1,2          H2      0,0,或 a
           STX    0,3(B)        H2        →B(R)
     A8R   LDX    0,1(LLINKO    G2      A8.单转动
           STX    0,4(RLINK)    G2      LINK(a,S)←LINK(-a,P)
           ST4    0,1(LLINK)    G2      LINK(-a,P)←S
     8H    STZ    0,1(B)        G       B(P)←0
     A10   CMP4   0,5(RLINK)    G       A10.最后一步
           JNE    *+3           G       如果 RLINK(T)≠S 则转移
           ST1    0,5(RLINK)    G3      RLINK(T)←P
           JMP    DONE          G3      转出
           ST1    0,5(LLINK)    G4      LLINK(T)←P
           JMP    DONE          G4      转出
           COM    +1
     T1    CON    0                     (3)的表
     T2    CON    0
           CON    -1
     6H    ENTX   +1            J2      rX←+1
     7H    STX    0,4(B)        J       B(S)←a
           LDX    HEAD(LLINK)   J       LLINK(HEAD)
           INCX   1             J             +1
           STX    HEAD(LLINK)   J         →LLINK(HEAD)
     DONE  EQU    *             1-S     插入完成 ▌
```

对平衡树插入的分析 [对数学不感兴趣的读者,请跳到(10)] 为了计算出算法 A 的运行时间,我们将要知道下列问题的答案:

- 在这个查找期间,作了多少次比较?
- 节点 S 和 Q 将相距多远?(换句话说,在步骤 6 中需要作多少调整?)
- 我们要如何经常地做单转动或双转动?

使用定理 A 来推导最坏情况运行时间的上限并不困难,但是当然在实践中我们需要知道平均的特性。由于这个算法看起来十分复杂,因此关于平均特性的理论确定还没有成功地完成,但已经得到了若干有趣的理论和经验的结果。

首先我们可以问及有关有 n 个内节点和高度为 h 的平均二叉树的个数 B_{nh}。对于小的 h,从关系式

$$B_0(z)=1,\ B_1(z)=z,\ B_{h+1}(z)=zB_h(z)(B_h(z)+2B_{h-1}(z)) \qquad (5)$$

不难计算生产函数 $B_h(z)=\sum_{n\geqslant 0}B_{nh}z^n$（参见习题 6）。

于是

$B_2(z)=2z^2+z^3$

$B_3(z)=\qquad 4z^4+6z^5+4z^6+z^7$

$B_4(z)=\qquad 16z^7+32z^8+44z^9+\cdots+8z^{14}+z^{15}$

而且，对于 $h\geqslant 3$，一般地说，$B_h(z)$有下列形式

$$2^{F_{h+1}-1}z^{F_{h+2}-1}+2^{F_{h+1}-2}L_{h-1}z^{F_{h+2}}+\text{复杂的项}+2^{h-1}z^{2^h-2}+z^{2^h-1} \qquad (6)$$

其中 $L_k=F_{k+1}+F_{k-1}$。（这个公式推广了定理 A。）有高度 h 的平衡树的总数是 $B_h=B_h(1)$。它满足递推式：

$$B_0=B_1=1 \quad B_{h+1}=B_h^2+2B_hB_{h-1} \qquad (7)$$

使得 $B_2=3$，$B_3=3\cdot 5$，$B_4=3^2\cdot 5\cdot 7$，$B_5=3^3\cdot 5^2\cdot 7\cdot 23$；而且，一般地

$$B_h=A_0^{F_h}A_1^{F_{h-1}}\cdots A_{h-1}^{F_1}A_h^{F_0} \qquad (8)$$

其中 $A_0=1$，$A_1=3$，$A_2=5$，$A_3=7$，$A_4=23$，$A_5=347$，…，$A_h=A_{h-1}B_{h-2}+2$。序列 B_h 和 A_h 非常快速地增长；事实上，它们是双倍指数的：习题 7 表明，存在一个实数 $\theta\approx 1.43687$ 使得

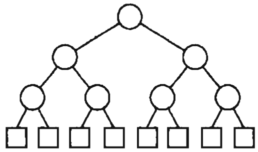
$$B_h=\lfloor\theta^{2^h}\rfloor-\lfloor\theta^{2^{h-1}}\rfloor+\lfloor\theta^{2^{h-2}}\rfloor-\cdots+(-1)^h\lfloor\theta^{2^0}\rfloor \qquad (9)$$

如果我们把 B_h 树的每一个都认为是同等可能的，则习题 8 证明在高度为 h 的一株树中节点的平均个数为

$$B'_h(1)/B_h(1)\approx(0.70118)2^h-1 \qquad (10)$$

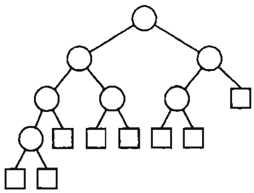
这表示具有 N 个节点的一株平衡树的高度通常比起接近 $\log_\phi N$ 来要更接近于 $\log_2 N$ 得多。

不幸的是，这些结果实际上和算法 A 没有多大关系，因为该算法的机制使得某些树比起其它树更有可能得多。例如，考虑 $N=7$ 的情况，其中 17 株平衡树是可能的。有 7！=5040 个可能的次序，其中可以插入 7 个键码，而且有 2160 次得到如下完全平衡的“完备树”。



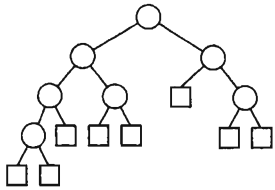
(11)

相反,斐波那契树



(12)

仅出现 144 次,而类似的树



(13)

出现 216 次。以任意四节点的平衡树来代替(12)和(13)的左子树,然后反射左和右,产生 16 种不同的树;由(12)产生的八种每个出现 144 次,而由(13)产生的那些每个出现 216 次。令人惊讶的是,(13)竟比(12)更为普遍。

完全平衡树以这样高的概率被得到这一事实,对应于相同概率的情况时的(10)一起,使得以下断言似真,即对于一株平衡树的平均查找时间应该是大约 $\lg N+c$ 个比较,其中 c 是某个小的常数。但是 R.W.Floyd 已经发现了 $\lg N$ 的系数不大可能为精确的 1,因为这株树的根将接近于中间。而它的两株子树的根将接近于四分之一的点;于是单转动和双转动不能容易地使根保持接近中间。经验测试指出,除非当 N 很小时,为插入第 N 个项所需要的真正平均比较次数为 $1.01\lg N+0.1$。

为了研究算法 A 的插入和重新平衡阶段的特性,我们可以像在图 23 中所示那样对平衡树的外节点进行分类。由一个外节点导出的通路可通过"+"和"-"的一个序列来确定(+表示一个右链接,-表示一个左链接),我们写下链接的描述直到达到有一个非空平衡因子的头一个节点为止。或者如果没有这样的节点,就直到达到根为止。然后根据当把一个内节点插入给定位置时,新的树将被平衡或不被平衡,我们写 A 或 B。因此从 [3] 开始的通路是 + + - B,意思是"右链接,右链接,左链接,不平衡"。以 A 结尾的一个描述在插入一个新节点之后不要求重新平衡;以 + + B 或 - - B 结尾的一个描述要求单个转动;以 + - B 或 - + B 结尾的一个描述要求一个双转动。当有 k 个链接出现在描述中时,步骤 A6 要调整 k - 1 个平衡因子。因此描述给出了支配步骤 A6 到 A10 的运行时间的实质性的事实。

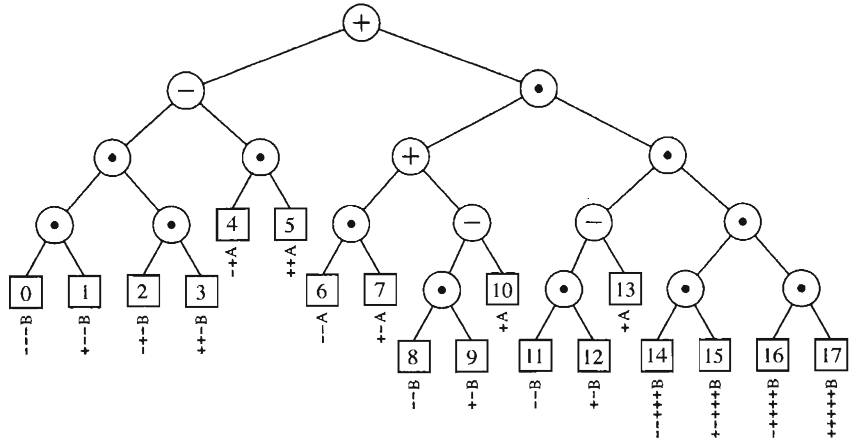


图 23 在插入之后描述算法 A 的特性的分类代码

表 1 插入第 N 项的近似概率

通路长度 k	无重新平衡	单转动	双转动
1	.143	.000	.000
2	.152	.143	.143
3	.092	.048	.048
4	.060	.024	.024
5	.036	.010	.010
>5	.051	.009	.008
平均 2.78	总共 .534	.233	.232

对于 $100 \leqslant N \leqslant 2000$ 的随机数的经验测试给出了对于各种类型的通路在表 1 中示出的近似概率;显然,当 $N \to \infty$ 时,这些概率迅速地趋向极限值。表 2 给出当 $N = 10$ 时对应于表 1 的精确概率,并且把输入的 10! 个排列看做是同等可能的。(对于所有 $N \geqslant 7$,在表 1 中作为".143"出现的概率实际上等于 1/7;参见习题 11。当 $N \leqslant 15$ 时,单转动和双转动是同等可能的,但当 $N \geqslant 16$ 时双转动稍微不太经常出现)。

由表 1,我们可以看出 $k \leqslant 2$ 时的概率大约是 $.143 + .153 + .143 + .143 = .582$;因此,步骤 A6 几乎在 60% 的时间里是十分简单的。在该步中平衡因子从 0 变成 ± 1 的平均次数大约是 1.8。在从步骤 A7 到 A10 的过程中平衡因子从 ± 1 变成 0 的平均次数近似于 $.534 + 2(.233 + .232) \approx 1.5$;因此,平均说来,插入一个新节点增加大约 $1.8 - 1.5 = 0.3$ 不平衡的节点。这和下列事实是相一致的,即在由算法 A 所构造的随机数中,在所有节点中,有 68% 被发现是平衡的。

表 2 插入第 10 项的精确概率

通路长度 k	无重新平衡	单转动	双转动
1	1/7	0	0
2	6/35	1/7	1/7
3	4/21	2/35	2/35
4	0	1/21	1/21
平均 247/105	53/105	26/105	26/105

C.C.Foster[*Prc. ACM Nat. Conf.* **20** (1965),192～205]已经提出算法 A 的特性的一个近似模型。这个模型不是严格地精确的,但对于给出某些启示说来,它是足够正确的。假定 p 是由算法 A 构造的一株大的树中一个给定节点的平衡因子为 0 的概率。于是,平衡因子为 +1 的概率是 $\frac{1}{2}(1-p)$,它为 −1 的概率同样是 $\frac{1}{2}(1-p)$。我们进一步假定(但没有论证)所有节点的平衡因子都是独立的。于是,步骤 A6 恰好置 $k-1$ 个平衡因子为非 0 的概率为 $p^{k-1}(1-p)$,所以 k 的平均值是 $1/(1-p)$。我们需要转动树的一部分的概率是 $q\approx 1/2$。平均说来,插入一个新节点将增加平衡节点的个数 p 个;这个数在步骤 A5 中实际上是 1,在步骤 A6 中是 $-p/(1-p)$,在步骤 A7 中是 q,在步骤 A8 和 A9 中是 $2q$,所以我们有

$$p = 1 - p/(1-p) + 3q \approx 5/2 - p/(1-p)$$

对 p 求解产生出和表 1 相当一致的结果:

$$p \approx \frac{9-\sqrt{41}}{4} \approx 0.649; \qquad 1/(1-p) \approx 2.851 \tag{14}$$

程序 A 的查找阶段(行 01-19)的运行时间是

$$10C + C1 + 2D + 2 - 3S \tag{15}$$

其中 C, C_1, S 和本章以前的诸算法一样,而 D 是在查找通路上遇到的不平衡节点的个数。经验测试表明,我们可以取 $D\approx\frac{1}{3}C$, $C1\approx\frac{1}{2}(C+S)$, $C+S\approx 1.01\lg N+0.1$,所以平均的查找时间近似于 $11.3\lg N+3-13.7S$ 单位。(如果查找的执行要比插入经常得多,我们当然可以对查找使用一个分开的更快的程序,因为那将没有必要来考察平衡因子;于是对于一次成功的平均运行时间将仅仅为大约$(6.6\lg N-3.4)u$,而且最坏情况的运行时间事实上将比由程序 6.2.2T 得到的平均运行时间还好些)。

当查找不成功时,程序 A 的插入阶段(行 20-45)的运行时间。为 $8F+26+(0,1$ 或 $2)$个单位。表 1 的数据表示,平均说来,$F\approx 1.8$。我们是否增加总的高度,或者不作重新平衡就转出,或做一个单转动或双转动,决定重新平衡阶段(行 46-101)花费 16.5,8,27.5 或 45.5(±0.5)个单位。第一种情况几乎决不出现,而其它情况以近似概率 .534,.233,.232 出现,所以程序 A 的组合插入一重新平衡部分的平均运

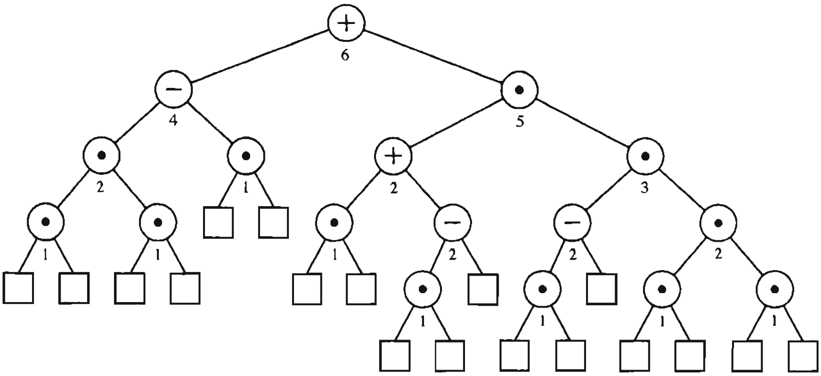


图 24　用于通过位置查找的 RANK 字段

行时间大约为 $63u$。

这些数字表明，在内存中对于一株平衡树的维护是相当快的，尽管这个程序相当长。如果输入数据是随机的，则 6.2.2 小节的简单树插入算法的每个插入快大约 $50u$；但是平衡树算法保证，即使对于非随机的输入数据，它仍然是可靠的。

程序 A 同程序 6.2.2T 的一个比较方法，是考虑后者的最坏情况。如果我们研究以递增的次序插入 N 个键码到开始时为空的一株树中所需要的时间，则程序 A 在 $N \leqslant 26$ 时比它慢，而在 $N \geqslant 27$ 时比它快。

线性表表示　现在我们转到本小节开始时所作的断言，即平衡树可以以这样一种方式来表示线性表，使得我们既可以快速地插入诸项（克服顺序分配的困难），也可以对表项目实施随机存取（克服链接分配的困难）。

这种想法就是在每个节点中引进一个新的称为 RANK 的字段，它指出该节点在其子树内的相对位置，即，1 加上它左子树中节点的个数。图 24 标出了图 23 的二叉树的 RANK 值。我们可以整个地消去 KEY 字段；或者，如果需要，也可以同时有 KEY 和 RANK 两个字段，使得有可能通过它们的键码值或者它们在表中的相对位置来查找项目。

利用这样一个 RANK 字段，通过位置进行查找，是对我们在一直研究的查找算法的一项直截了当的修改。

算法 B（通过位置进行树查找）　给定表示成一株二叉树的线性表，给定 k，本算法寻找该表的第 k 个元素（在对称次序下树的第 k 个节点）。假定二叉树有 LLINK 字段和 RLINK 字段，以及如算法 A 中那样的一个表头，加上如前所述的一个 RANK 字段。

B1.［初始化］置 M←k，P←RLINK(HEAD)。

B2.［比较］如果 P＝Λ，则此算法以失败告终。（仅当 k 大于树中的节点数，或 $k \leqslant 0$ 时，这才可能发生）。否则，如果 M＜RANK(P)，则转到 B3；如果 M＞RANK(P)，则转到 B4；如果 M＝RANK(P)，则本算法成功地结束（P 指向第 k 个

节点)。

B3. [左移] 置 P←LLINK(P)并返回 B2。

B4. [右移] 置 M←M－RANK(P)和 P←RLINK(P)并返回 B2。 ▌

在本算法中惟一有趣之点是在步骤 B4 中 M 的操作。我们可以以类似的方式修改插入步骤,尽管其细节是有些技巧的。

算法 C(平衡树按位置插入)　给定表示成一株平衡二叉树的线性表,并给定 k 和指向一个新节点的指针 Q,本算法恰在此表的第 k 个元素之前插入该新节点。如果 $k=N+1$,则这个新节点恰被插入到此表的最末元素之后。

假定二叉树是非空的并有 LLINK、RLINK 和 B 字段及一个表头,如算法 A 中那样,再加上如前所述的一个 RANK 字段。本算法仅是算法 A 的改写;其差别只是它使用和更新 RANK 字段而不是 KEY 字段。

C1. [初始化] 置 T←HEAD,S←P←RLINK(HEAD),U←M←k。

C2. [比较] 如果 M≤RANK(P),则转到 C3,否则转到 C4。

C3. [左移] 置 RANK(P)←RANK(P)＋1(我们将插入一个新节点到 P 的左边)。置 R←LLINK(P)。如果 R＝Λ,则置 LLINK(P)←Q,并转到 C5。否则如果 B(R)≠0,则置 T←P,S←R 以及 U←M。最后,置 P←R 并返回到 C2。

C4. [右移] 置 M←M－RANK(P)和 R←RLINK(P)。如果 R＝Λ,则置 RLINK(P)←Q 并转到 C5。否则如果 B(R)≠0,则置 T←P,S←R 以及 U←M。最后,置 P←R 并返回 C2。

C5. [插入] 置 RANK(R)←1,LLINK(Q)←RLINK(Q)←Λ,B(Q)←0。

C6. [调整平衡因子] 置 M←U。(这恢复了当 P 是 S 时 M 以前的值;所有 RANK 字段现在都已适当地赋值)。如果 M＜RANK(S),则置 R←P←LLINK(S),而且 a←－1;否则置 R←P←RLINK(S),a←＋1,和 M←M－RANK(S)。然后重复做下列操作直到 P＝Q 为止:如果 M＜RANK(P),则置 B(P)←－1 以及 P←LLINK(P);如果 M＞RANK(P),则置 B(P)←＋1 和 M←M－RANK(P)及 P←RLINK(P)。(如果 M＝RANK(P),则 P＝Q 而且进到下一步)。

C7. [平衡动作] 现在出现若干情况:

i)如果 B(S)＝0,则置 B(S)←a,LLINK(HEAD)←LLINK(HEAD)＋1,并结束此算法。

ii)如果 B(S)＝－a,则置 B(S)←0 并结束此算法。

iii)如果 B(S)＝a,则如果 B(R)＝a 即转到步骤 C8,如果 B(R)＝－a 即转到步骤 C9。

C8. [单转动] 置 P←R,LINK(a,S)←LINK(－a,R),LINK(－a,R)←S,B(S)←B(R)←0。如果 a＝＋1,则置 RANK(R)←RANK(R)＋RANK(S);如果 a＝－1,则置 RANK(S)←RANK(S)－RANK(R)。转向 C10。

C9. [双转动] 进行步骤 A9(算法 A)的所有操作。然后,如果 a＝＋1,则置 RANK(R)←RANK(R)－RANK(P),RANK(P)←RANK(P)＋RANK(S);如果 a＝－1,

则置 RANK(P)←RANK(P)+RANK(R),然后 RANK(S)←RANK(S)−RANK(P)。

C10. [最后一步] 如果 S=RLINK(T)则置 RLINK(T)←P,否则置 LLINK(T)←P。▮

* **删去、连接等** 对于平衡树和维持平衡还可做许多其它事情,但这些算法都是十分冗长的,它们的细节已超出了本书的范围。这里我们只讨论一般的思想,有兴趣的读者不难自己补充其细节。

如果采用正确的方法,则删去的问题可以在 $O(\log N)$步内解决[C.C.Foster, "A Study of AVL Trees", Goodyear Aerospace Corp. report GER-12158(1965 年 4 月)]。首先,我们可以把任意节点的删去简单地归结为一个节点 P 的删去,对这个节点 P 来说,如同在算法 6.2.2D 中那样,LLINK(P)或 RLINK(P)是 Λ。这个算法还应加以修改使它造一张确定到节点 P 的通路的指针表,即

$$(P_0, a_0), (P_1, a_1), \cdots, (P_l, a_l) \tag{16}$$

其中,P_0=HEAD,$a_0=+1$;对于 $0 \leqslant i < l$,LINK(a_i, P_i)$=P_{i+1}$,$P_l=P$;而且 LINK(a_l, P_l)$=\Lambda$。当我们往下查找树时,这张表可放置在一个辅助栈上。删去节点 P 的过程置 LINK(a_{l-1}, P_{l-1})←LINK($-a_l, P_l$),而且我们必须调整节点 P_{l-1}处的平衡因子,因为这个节点的 a_k 子树的高度刚刚减少了;应该使用下列调整方法:如果 $k=0$,则置 LLINK(HEAD)←LLINK(HEAD)−1 并结束此算法,因为整个树高度减少了。否则考察平衡因子 B(P_k);有三种情况:

i) B(P_k)$=a_k$。置 B(P_k)←0,k 减 1,并对 k 的新值重复此调整方法。

ii) B(P_k)=0。置 B(P_k)为 $-a_k$ 并结束删去算法。

iii) B(P_k)$=-a_k$。需要重新平衡!

需要重新平衡的这些情况,几乎同我们在插入算法中遇到的一样;再次参考(1),A 是节点 P_k,B 是节点 LINK($-a_k, P_k$),它在与出现删去的分支相反的分支上。惟一的新特征是节点 B 可能是平衡的;这导致一个新的情况 3,它和情况 1 类似,但 β 的高度是 $h+1$。在情况 1 和 2 中,如同在(2)中那样的重新平衡意味着减少高度,所以我们置 LINK(a_{k-1}, P_{k-1})为(2)的根,k 减 1,并对这个新的 k 值重新开始调整。在情况 3 中,我们做一个单转动,这就保持了 A 和 B 两者的平衡因子非 0,而不改变整个高度;因此,在使 LINK(a_{k-1}, P_{k-1})指向节点 B 之后,我们就结束这个算法。

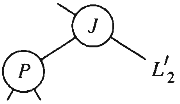
删去和插入之间的重要差别是:删去可能需要多达 $\log N$ 次转动,而插入所需的转动决不多于一次。如果我们试图删去一株斐波那契树最右的节点(见 6.2.1 小节图 8),那么,这个原因就显得很清楚了。但是经验测试表明,每次删去平均只需作 0.21 次转动。

对于线性表表示使用平衡树,也提示了需要一个*连接算法*,以便把整株树 L_2 插入到树 L_1 的右边,而不破坏平衡。Clark A Crane,已经首先想出了用于连接的漂亮算法:假设高度(L_1)≥高度(L_2);其它情况是类似的。删去 L_2 的头一个节点,称它为*交界节点* J,并令 L_2' 代表新树 $L_2 \setminus \{J\}$。现在顺着 L_1 的诸右链接往下走,直到

到达一个节点 P，使得

$$高度(P) - 高度(L_2') = 0 或 1$$

为止；这总是可能的，因为每往下走一级，高度就改变 1 或 2，然后以



来代替，进而调整 L_1，就好像新节点 J 是刚由算法 A 插入的那样。

Crane 还解决了更困难的求逆问题，把一个表分成为两部分，使得它们的连接还是原来的表。例如，考虑把图 20 中的表分开来得到两个表，一个含{A，…，I}，另一个含{J，…，Q}；这需要对于诸子树作大量的重组工作。一般地说，当我们需要在某个给定的节点 P 处来分开一株树时，到 P 的通路有些像图 25 中那样。我们希望构造一株左树，它以对称次序包含 $\alpha_1, P_1, \alpha_4, P_4, \alpha_6, P_6, \alpha_7, P_7, \alpha, P$ 诸节点，以及一株右树，它包含 $\beta, P_8, \beta_8, P_5, \beta_5, P_3, \beta_3, P_2, \beta_2$。这可以通过一系列的连接来做：首先在 α 的右边插入 P，然后利用 P_8 作为交界节点连接 β 和 P_8，利用 P_7 作为交界节点连接 α_7 和 αP，利用 P_6 连接 α_6 和 $\alpha_7 P_7 \alpha P$，利用 P_5 连接 $\beta P_8 \beta_8$ 和 β_5，等等；在到 P 的通路上的节点 $P_8, P_7, \cdots, P_1$ 均被用作交界节点。Crane 已经证明，当原来的树包含 N 个节点时，这个分开算法仅花费 $O(\log N)$ 个时间单位；实际的原因是利用一个给定的交界点的连接要花费 $O(k)$ 步，其中 k 是正被连接的树之间的高度差，而且必须累加起来的 k 的值实质上形成了正被构造的左树和右树两者的一个收缩的级数。

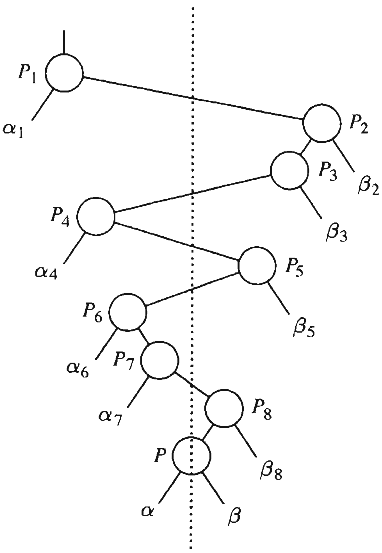


图 25 分开一个表的问题

所有这些算法都可以利用 KEY 或 RANK 字段或者两者使用，尽管在连接的情况下，L_2 的键码必须全都大于 L_1 的键码。为了通用的目的，通常可取的是使用一株三重链接的树，它具有 UP 链以及 LLINK 和 RLINK，连同一个新的一位二进位字段，这个字段确定一个节点是其父亲的左儿子还是右儿子。三重链接树表示稍微简化了这些算法，并且使我们有可能确定树中的节点而无需明确地跟踪通到该节点的通路；给定 P，我们可以写出一个子程序删去 NODE(P)，或者删去在对称次序下跟随 NODE(P)的节点或者求出包含 NODE(P)的表，等等。在三重链接树的删去算法中不必构造表(16)，因为 UP 链为我们提供了所需要的信息。当然，当插入、删去和转动正被实施时，一株三重链接树要求改变稍微多的链接。使用一株三重链接树来代替一株双重链接树，类似于使用两路链接代替一路链接的情形：我们可以从任意点开始，

或向前进行或向后进行。以三重链接平衡树为基础的表算法的完整的描述，出现在 Clark A Crane 的博士论文中(斯坦福大学，1972)。

AVL 树的一些选择　许多组织树的其它方法已被提出，以保证对数阶的存取时间。例如，C.C.Foster[*CACM* **16** (1973)，513～517] 考虑了当我们允许子树的高度差至多为 k 时出现的二叉树。这样的结构可称做 HB[k]树(意思是“高度平衡”)，而通常的平衡树表示其特殊情况 HB[1]。

加权平衡树的有趣概念，已经为 J.Nievergelt，E.Reingold 和黄泽权所研究。不去考虑树的高度，他们约定所有节点的子树必须满足

$$\sqrt{2}-1<\frac{\text{左权}}{\text{右权}}<\sqrt{2}+1 \tag{17}$$

其中，左权和右权分别累计左子树和右子树中外节点的数目。有可能证明，仅仅利用如算法 A 中进行重新平衡的单转动和双转动，在插入之下仍能维持权的平衡。(参见习题 25)。然而，在一次插入期间可能需要做许多重新平衡。在增加查找时间的条件下，减少重新平衡的次数，有可能放松(17)的条件。

乍一看，权平衡树似乎比以前单纯的平衡树需要更多的内存。但是事实上，它们有时要求反倒稍少些！对于线性表表示，如果每个节点已有一个 RANK 字段，则这恰巧是左权。而当我们沿树下移时，有可能记住对应的右权。然而，为了维持权平衡所需要的管理操作，看来要花费比算法 A 更多的时间，而每个节点省去两位二进数位，似乎还抵不上所增加的麻烦。

Why don't you pair 'em up in threes?
为什么你不把它们配对成三个呢？
——*归属于 YOGI BERRA (大约 1970 年)*

AVL 树的另一种有趣的选择，称做“2-3 树”，它是由 John Hopcroft 于 1970 年提出的[见 Aho，Hopcroft 和 Ullman，*The Design and Analysis of Computer Algorithms* (Mass：Addison-Wesley，1974)，第 4 章]。这个思想要求在每个节点处作 2 路分支或 3 路分支，并约定所有外节点都出现在相同的级上。每个内节点包含一个或两个键码，如图 26 所示。

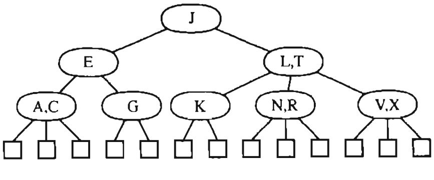


图 26　一株 2-3 树

插入到一株 2-3 树要比插入到一株平衡树易于说明：如果我们要把一个新的键码插入到仅含一个键码的节点中，则只要简单地把它插入作为第二个键码。另一方面，如果这个节点已经含有两个键码，则我们就把它分为两个各有一个键码的节点，而把中间的键码插入到父母节点处。如果父母节点已有两个键码，这又可以引起父母节点以相同的方式被划分。图 27 示出了把一个新的键码插入到图 26 的 2-3 树的过程。

Hopcroft 发现，正像在 AVL 树的对应操作那样，对于 2-3 树也可以相当直截了当的方式进行删去、连接和分开等操作。

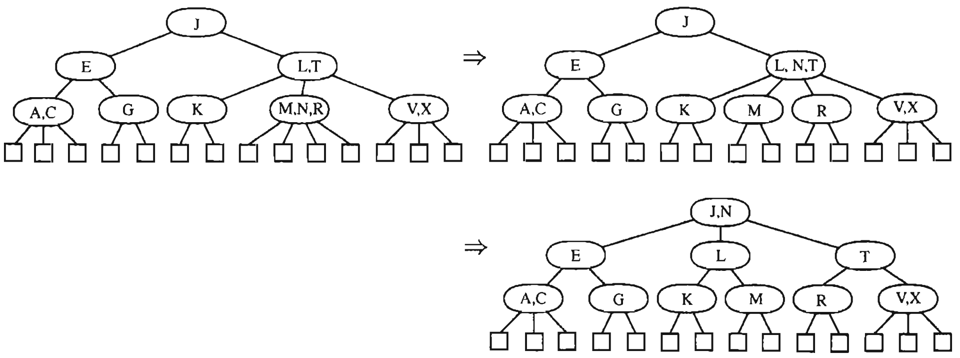


图 27 把新键码"M"插入图 26 的 2-3 树中

R.Bayer[*Proc. ACM-SIGFIDET Workshop* (1971),219～235]已经提议使用 2-3 树的一种有趣的二叉树表示。见图 28,它给出图 26 的二叉树表示;在每个节点中,用一个二进位来区分诸"水平"的 RLINK 和诸"垂直"的 RLINK。注意,正如任何二分查找树那样,树的键码以对称方式从左到右出现。结果表明,当我们如图 27 那样插入一个新键码时,在这样一株二叉树上需要实施的转换,恰恰是把一个新键码插入到一株 AVL 树时所用的单转动和双转动,不过我们只需要一个单转动和一个双转动,而不是在算法 A 和 C 中的左右反演。

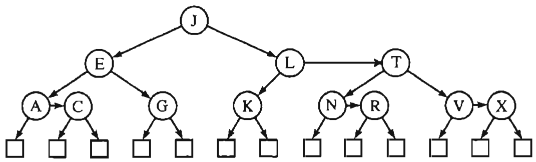


图 28 图 26 的 2-3 树表示成一株二分查找树

对这些思想的精心推敲已经导致许多另外风味的平衡树,最著名的有红黑树,也叫做对称二叉 B 树或半平衡树[R.Bayer,*Acta Informatica* **1** (1972),290～306;L.Guibas 和 R.Sedgewick ,*FOCS* **19** (1978),8～21;H.J.Olivie,*RAIRO Informatique Theorique* **16** (1982),51～71;R.E.Tarjan,*Inf, Proc. Letters* **16**(1983),253～257;T.H.Cormen,C.E.Leiserson 以及 R.L.Rivest,*Introduction to Algorithms* (麻省理工学院出版社,1990),第 14 章;R.Sedgewick,*Algorithms in C* (Addison-Wesley, 1997),§13.4]。有一个称为歇斯底里的 B 树或(a,b)-树的很密切关联的族。主要是(2,4)-树[D.Maier 和 S.C.Salveter *Inf. Proc. Letters* **12**(1981),199～202;S.Huddleston 和 K Mehlhorn,*Acta Informatica* **17**(1982),157～184]。

当对一些键码的访问要比对其它键码的访问更频繁得多时,如同在 6.2.2 小节中最优二分查找树一样,我们要使这些重要的键码相对地靠近根。S.W.Bent, D.D.Sleator 和 R.E.Tarjan, *SICOMP* **14** (1985), 545～568; J . Feigenbanm 和

R.E.Tarjan,*Bell System Tech J.* **62** (1983),3139～3158 已经提出了称为有偏向树的动态树,这种树使得有可能在最优的一个常数因子的范围内维持加权的平衡。然而,这些算法十分复杂。

基于和 6.1 节中讨论的移向前面和转换位置的启发或探索相类似的思想,随后 D.D.Sleator 和 R.E.Tarjan 提出了称为倾斜树的简单得多的自调整数据结构[*JACM* **32**(1985),652～686];在此之前,B.Allen 和 I.Mnuro[*JACM* **25**(1978),526～535]和 J.Bitner[*SICOMP* **8**(1979),82～110]已经剖析了类似的技术。倾斜树和上面已经提到过的其它类型的平衡树一样,既支持插入和删去,也支持连接和分开操作,而且是以一种特别简单的方式进行的。而且,当对任何操作序列平摊时,已知为访问在一个倾斜树中的数据所需要的时间至多是对于静态最优树的访问时间的一个小的常数倍。Sleator 和 Tarjan 猜测,对倾斜树总共的访问时间至多是通过无论什么样的二叉树算法来访问数据和动态地实施转动的最优时间的一个常数倍。

随机化导致了一些这样的方法,它们看起来甚至比倾斜树更简单、更快。Jean Vuillemin [*CACM* **23** (1980),229～239]引进了笛卡儿树,其中每个节点有两个键码 (x,y)。x 部分如同在二分查找树中一样是自左至右有序的;y 部分如同在 5.2.3 小节的优先队列树一样是由顶向下有序的。C.R.Aragon 和 R.G.Seidel 对这个数据结构起了个更有色彩的名称树堆(treap)。因为它很利落地把树的概念和堆的概念组合在一起。精确地说,如果诸 x 和诸 y 都不同,则一个树堆可以通过 n 个给定的键码对 $(x_1,y_1),\cdots,(x_n,y_n)$ 构造出来。得到它的一个方法是按照 y 的次序,通过算法 6.2.2T 插入诸 x;但还有一个简单的算法,它把任何新的键码直接插入到任何树堆中。Aragon 和 Seidel [*FOCS* **30**(1989),540～546] 发现,如果诸 x 是通常的键码而诸 y 是随机选择的,则我们可以确信,树堆有一个随机二分查找树的形状。特别是,具有随机 y 值的一个树堆将总是相当好地平衡的,但有指数上很小的概率(参见习题 5.2.2-42)。Aragon 和 Seidel 还证明,树堆能很容易地成为有偏向的,使得比如说,当具有相对频率 f 的一个键码 x 同 $y=U^{1/f}$ 相关联时,则这个键码将稳定地接近根而出现,其中 U 是 0～1 之间的一个随机数。由 D.E.Knuth 所作的同凸外壳的计算有关的一些实验中[*Lecture Notes in Comp.Sci.* **606**(1992),53～55]。树堆的性能一致地比倾斜树更好些。

本书的下一版本计划把新的 6.2.5 小节专用于讨论随机化的数据结构,它将讨论"跳越表"[W.Pugh, CACM ***33****(1990),668～676)]以及"随机化的二分查找树"[S.Roura 和 C.Martinez, JACM* ***45****(1998),288～323],以及树堆。*

习　题

1.[*01*] 在(1)的情况 2 中,为什么通过简单地交换 A 和 B 的左子树来恢复平衡并不是一种好的想法?

2.[*16*] 如果我们以 B(S)＝0 达到步骤 A7,试说明为什么这株树会高出一级来。

▶3.[*M25*] 证明:具有 N 个内部节点的一株平衡树决不包含多于 $(\phi-1)N\approx0.61803N$ 个其

平衡因子非 0 的节点。

4.[*M22*] 证明或否定:在具有 $F_{h+1}-1$ 个内部节点的所有平衡树当中,阶数为 h 的斐波那契树有最大的内部路径长度。

▶5.[*M25*] 证明或否定:如果使用算法 A 以递增次序逐次地插入链 $K_2,\cdots,K_N$ 到开始时仅含单个键码 K_1 的一株树中,其中 $K_1<K_2<\cdots<K_N$,则所产生的树总是最优的(即在所有 N 节点的二叉树中,它有极小内部路径长度)。

6.[*M21*] 证明等式(5)定义了高度为 h 的平衡树的生成函数。

7.[*M27*] (N.J.A.Sloane 和 A.V.Aho) 证明高度为 h 的平衡树个数的著名公式(9)。[提示:设 $C_n=B_n+B_{n-1}$,且利用如下事实,即对于很大的 n,$\log(C_{n+1}/C_n^2)$ 非常小。]

8.[*M24*] (L.A.Knizder)证明存在一个常数 β,使得当 $h\to\infty$ 时,$B'_h(1)/B_h(1)=2^h\beta-1+O(2^h/B_{h-1})$。

9.[*HM44*] 具有 n 个内节点的平衡二叉树的近似株数 $\sum_{h\geqslant 0}B_{nh}$ 等于多少?近似平均高度 $\sum_{h\geqslant 0}hB_{nh}/\sum_{h\geqslant 0}B_{nh}$ 等于多少?

▶10.[*27*] (R.C.Richards) 证明,一个平衡树的形状可以由它的平衡因子表 B(1)B(2)…B(N)以对称次序惟一地构造出来。

▶11.[*M24*] (Mark R.Brown) 证明当 $n\geqslant 6$ 时,在由算法 A 构造的 n 个内节点的一株随机平衡树中,其类型为 + A、- A、+ + B、+ - B、- + B、- - B 等的任何一种的外节点的平均个数恰为 $(n+1)/14$。

▶12.[*24*] 当把八个节点插入到一株平衡树中时,程序 A 可能的极大运行时间是多少?对于这个插入,可能的极小运行时间是多少?

13.[*05*] 为什么比较好的办法是如正文中定义的那样使用 RANK 字段,而不是简单地存储每个节点的下标作为它的键码(称第一个节点“1”,第二个节点“2”,等等)?

14.[*11*] 能否修改算法 6.2.2T 和 6.2.2D 使它们对线性表也适用,使用一个 RANK 字段,就如同对本小节的平衡树算法所进行的修改那样。

15.[*18*] (C.A.Crane) 假设一个有序线性表被表示为一株二叉树,且在每个节点处有 KEY 和 RANK 字段。试设计一个算法,它在树上查找一个给定的键码 K,并确定 K 在表中的位置;即,它求得数 m,使得 K 是第 m 个最小的键码。

▶16.[*20*] 利用正文中的删去算法,在从图 20 中同时删去节点 E 和根节点 F 之后,便得到一株平衡树。试画出这株树。

▶17.[*21*] 利用正文中建议的连接算法,把斐波那契树(12)连接到图 20 中树(a)右边,(b)左边之后,便得到两株平衡树。试画出这两株树。

18.[*22*] 利用正文中所建议的分开算法,把图 20 分成两部分{A,…,I}和{J,…,Q}之后便得到两株平衡树。试画出这两株树。

▶19.[*26*] 找出一个方法,转换一株给定的平衡树,使得在根处的平衡因子不是 - 1。你的转换应该保持节点的对称次序;而且不管原有树的大小如何,它都应该在 $O(1)$ 的时间单位中产生另一株平衡树。

20.[*40*] 剖析利用有限制的一类平衡树的思想,这些树的节点的平衡因子限于是 0 或 + 1。(于是 B 字段的长度可减小到一个二进数位)。对于这样的树是否有一个相当有效的插入过程?

▶21.[*30*] (完全平衡) 试设计一个算法,它构造习题 5 的意义下是最优的 N 个节点的二叉树。你的算法应该使用 $O(N)$ 步而且是“联机的”,即是说,它逐个地以递增的次序输入节点,并随即构造部分树,而不必预先知道 N 的最后的值。(当重新构造一株失去平衡的树时,或者当合并

两株树的键码成为一株树时,使用这样一个算法是适当的。)

22.[*M20*] 对于加权平衡树,类似于定理 A 的结果是什么?

23.[*M20*] (E.Reingold) 证明高度平衡树与加权平衡树之间没有简单的关系。(a)证明在(17)的意义下存在(左权)/(右权)为任意小比值的加权平衡树。(b)证明存在其左子树和右子树的高度差为任意大的加权平衡树。

24.[*M22*] (E.Reingold) 证明,如果我们把条件(17)增强到

$$\frac{1}{2} < \frac{\text{左权}}{\text{右权}} < 2$$

则满足这一条件的仅有的二叉树是具有 2^n-1 个内节点的完全平衡树(在这样的树中,左权和右权在所有节点处都完全相等)。

25.[*27*] (J.Nievergelt,E.Reingold,黄泽权) 证明有可能设计加权平衡树的插入算法,使得条件(17)成立,并且使得每个插入至多进行 $O(\log N)$次转动。

26.[*40*] 试剖析对于 $t>2$ 的平衡 t 叉树的性质。

▶27.[*M23*] 试估计在有 N 个内节点的 2-3 树中进行查找所需要的极大比较次数。

28.[*41*] 编制 2-3 树算法的有效实现方案。

29.[*M47*] 试分析在随机插入下,2-3 树的平均特性。

30.[*26*] (E.McCreight) 2.5 节讨论了动态存储分配的若干策略,包括"最好的适合"(从满足要求的所有那些区域当中选择尽可能小的可利用区域)和"第一个适合"(在所有满足要求的那些区域当中选择具有最小地址的可利用区域)。证明,如果以一种适当的方式把可利用空间链接在一起成为一株平衡树,则有可能只需要 $O(\log n)$个时间单位来进行(a)最好的适合和(b)第一个适合的分配,其中 n 是可利用区域数(在 2.5 节中给出的算法,花费阶为 n 步)。

31.[*34*] (M.L.Fredman 1975) 试想出具有如下性质的线性列表的一个表示——给定 m,在位置 $m-1$ 和 m 之间插入一个新节点花费 $O(\log m)$个时间单位。

32.[*M27*] 给定两个 n 节点的二叉树 T 和 T',如果通过一序列的 0 次或多次向右的转动,由 T 可得到 T',就说 $T \leqslant T'$。证明 $T \leqslant T'$ 当且仅当对于 $1 \leqslant k \leqslant n$, $r_k \leqslant r'_k$,其中 r_k 和 r'_k 表示在对称次序下 T 和 T' 的第 k 个节点的右子树分别的大小。

▶33.[*25*] (A.L.Buchsbaum) 说明怎样含蓄地对一个 AVL 树的平衡因子进行编码,这样当对此树进行访问时,以增加工作量为代价,每个节点节省两个二进位。

Samuel considered the nation of Israel, tribe by tribe,
and the tribe of Benjamin was picked by lot.
Then he considered the tribe of Benjamin, family by family,
and the family of Matri was picked by lot.
Then he considered the family of Matri, man by man,
and Saul son of Kish was picked by lot.
But when they looked for Saul he could not be found.
塞缪尔按部落一个一个地考虑以色列的民族,并且抽签选中了本杰明的部落。
然后他按家族一个一个地考察本杰明的部落,并且抽签选中马特里的家族。
然后他按人一个一个地考虑马特里的家族,并且抽签选中基斯的儿子索尔。
但当他们寻找索尔时,却找不到他。
—1 Samuel 10:20－21

6.2.4 多路树

我们一直在讨论的树查找方法主要用于内部查找,即所要考察的表是完全包含在计算机高速内存中的。下面我们考虑,当从一个非常大的文件中查找信息时进行外部查找的问题。这些非常大的文件,出现在直接存取存储装置,例如磁盘或磁鼓上(5.4.9 小节有关于磁盘和磁鼓的介绍)。

如果我们选择一种适当的方法来表示树,那么树结构本身很适合于外部查找。考虑图 29 中所示的大型二分查找树,并想像它已经存在一个磁盘文件中(树的 LLINK 和 RLINK 现在是磁盘地址而不是内存地址)。如果我们以一个朴实的方式查找这株树,简单地应用对内部树查找已经学过的算法,则大约要做 lg$N$ 次磁盘访问,才能完成查找。当 N 是 100 万时,这意味着我们需要做 20 次左右的寻找。但假如把这个表分成许多 7 个节点的"页",如图 29 中虚线所示;如果我们一次访问一页,则仅需要前述次数的三分之一,所以查找大约快三倍。

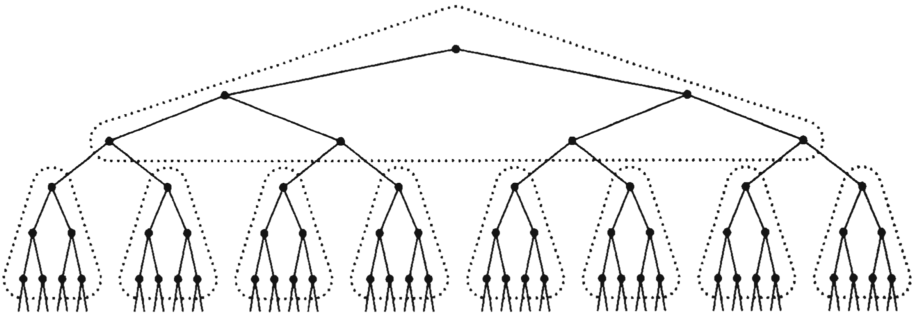


图 29 一株大型二分查找树可以分成许多"页"

以这一方式把节点组合成页,实际上把一株二叉树变成了八叉树,在每个页节点处有 8 路分支。如果我们让页更大一些,在每次磁盘访问之后有 128 路分支,则在仅仅寻找三页之后,我们就可以在一个百万键码的表中找出任何所希望的键码。根页可以常驻内存,因此只需对磁盘进行两次访问,而且在任一时刻,内存中的键码数都不超过 254 个。

当然我们不要使页任意大,因为,内存的大小有限,并且它需花费更长的时间来读入一个更大的页。例如,假设它花费 72.5 + 0.05$m$ms 来读允许 m 路分支的一个页,每页的内部处理时间约为 $a + b\lg m$,其中 a 相对 72.5ms 来说是很小的,因此,为查找一个大表所需要的时间总量近似地正比于 lg N 乘以

$$(72.5 + 0.05m)/\lg m + b$$

当 $m \approx 307$ 时,这个量实现极小值;实际上,极小值是非常"广"的,200～500 之间的所有 m 都接近于达到一个最优值。实践中,根据具体外部存储设备的特征,以

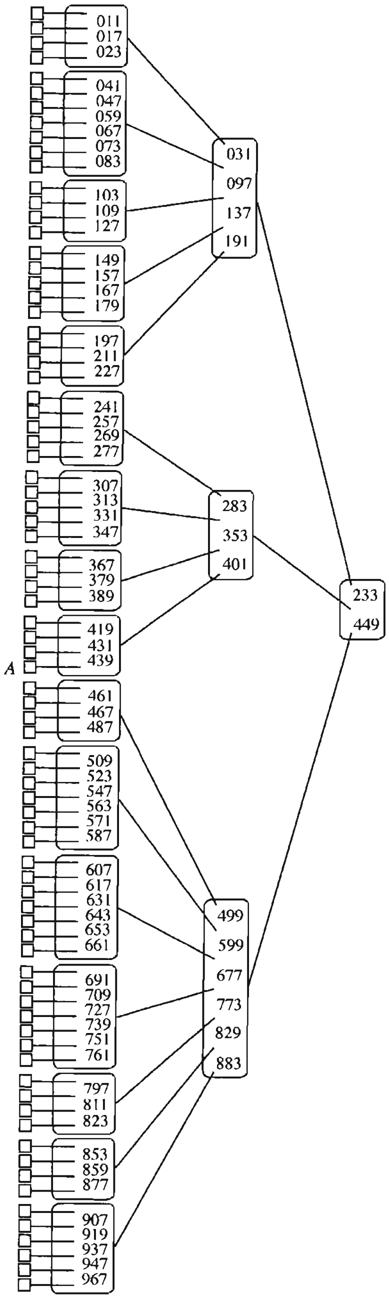


图 30　阶为 7 的一株 B 树，所有叶都在第 3 级上。每个节点包含 3，4，5 或 6 个键码。在键码 449 之前的叶已标记了 A；参见(8)

及表中记录的长度，将有一个类似的好的 m 值范围。

W.I.Landauer[*IEEE.Trans.* **EC-12**(1963),863～871]提议，在构造一株 m 叉树时，首先要使第 l 级接近充满，然后才能往第 $l+1$ 级上放东西。这个方案需要一个相当复杂的转动方法，因为可能需要对整株树作重大的改变才能插入一个新的项目；Landauer 假设，我们需要比插入或删去项目更为经常得多地查找树中的项目。

当一个文件存在磁盘上，且插入或删去不多时，则一株三极树是适当的。其中分支的第一级确定使用什么柱面，分支的第二级确定在该柱面上的适当的磁道，第三级包含记录本身。这个方法称为*索引顺序文件组织*[参见 *JACM* **16**(1969),569～571]。

R.Muntz 和 R.Uzgalis[*Proc. Princeton Conf. on Inf. Sciences and Systems* **4** (1970),345～349]已经提议修改树查找和插入方法，即算法 6.2.2T，使得只要可能，就让所有插入都对那些与其父亲节点属于相同页的节点进行；如果该页已满，则只要可能，就开始一个新的页。如果页数无限，且如果数据以随机的次序到达，则可以证明，平均页存取次数近似为 $H_N/(H_m-1)$，仅比我们在最好的 m 叉树中所得到的稍微多一点(见习题 18)。

B 树 1970 年 R.Bayer 和 E.McCreight 已经发现了借助多路树分支研究外部查找的一个新方法[*Acta Informatica* **1**(1972),173～189]，而且大约在同一时间 M.Kaufman 也独立地发现了同一方法[未发表]。他们的思想，是以一类称为 B 树的多方面适用的新型数据结构为基础的，它使得在最坏的情况下，仍有可能利用比较简单的算法，以“有保证的”效率来查找和更新一个大型文件。

阶 m 的一株 B 树，是满足下列性质的一株树：

i) 每个节点至多有 m 个儿子。

ii) 除了根和叶之外，每个节点的儿子个数至少是 $m/2$。

iii) 根至少有两个儿子(除非它是一片叶)。

iv) 所有叶都出现在同一级上，而且不带信息。

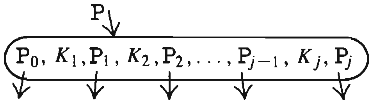
v) 具有 k 个儿子的非叶节点含有 $k-1$ 个键码。

(通常，一片“叶”是一个终端节点，它没有儿子。因为叶不带有信息，故我们可以把它们看做实际上不在树内的外部节点，因而 Λ 是指向一片叶的指针。)

图 30 示出阶 7 的一株 B 树，每个节点(除根和叶外)都有介于$\lceil 7/2\rceil$和 7 之间的儿子，所以它包含 3、4、5 或 6 个键码。允许根节点包含 1～6 个键码；在现在情况下，它有 2 个键码。所有的叶都在级 3 上。注意，(a)键码从左到右以递增的次序出现，这是对称次序概念的一种自然的推广；(b)叶的片数恰比键码的个数大 1。

显然，我们对阶 1 或 2 的 B 树没有兴趣，所以仅仅考虑 $m\geqslant 3$ 的情形。在接近 6.2.3 小节末尾处定义的 2-3 树等价于阶 3 的 B 树。(Bayer 和 McCreight 仅考虑 m 是奇数的情况。有些作者把阶为 m 的 B 树当作我们所说的阶 $2m+1$ 的 B 树)。

包含 j 个键码和 $j+1$ 个指针的一个节点可以表示作



(1)

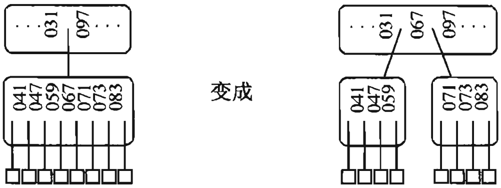
其中 $K_1<K_2<\cdots<K_j$，且 P_i 指向包含 K_i 和 K_{i+1}之间的键码的子树。因此在一株 B 树中的查找十分直截了当：在节点(1)已经进入内存之后，在键码 $K_1,K_2,\cdots,K_j$ 当中查找给定的变元（当 j 很大时，大概作一个二分查找；但当 j 很小时，顺序查找是最好的）。如果查找成功，则我们已经找到了所希望的键码；但如果由于变元位于 K_i 和 K_{i+1}之间而使查找不成功，则取由 P_i 指出的节点，并继续这个过程。如果变元小于 K_1，则使用指针 P_0，如果变元大于 K_j，则使用 P_j。如果 $\mathrm{P}_i=\Lambda$，则查找是不成功的。

B 树的优点是插入也十分简单。例如，考虑图 30，每片叶对应一个新的插入可能发生的位置。如果我们要插入新的键码 337，则只需把适当的节点从



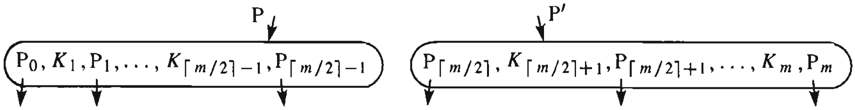
(2)

另一方面，如果我们要插入新的键码 071，就没有空间了，因为在级 2 上的对应节点已经“满”了。这种情况可以通过把节点分成两部分来处理，每部分有三个键码，并把中间的键码送到级 1 上：



(3)

一般说来，如果我们要把一新项目插入到阶为 m 的 B 树中去，当所有叶都在级 l 上时，则把新的键码插入到级 $l-1$ 的适当的节点上。如果该节点现在含 m 个键码，使得它有形式(1)，且 $j=m$，则把它分成两个节点



(4)

并且把键码 $K\lceil m/2\rceil$插入到原来节点的父节点处。（于是，父节点中的指针 P 为序列 $\mathrm{P},K_{\lceil m/2\rceil},\mathrm{P}'$所代替）。这个插入可能引起父节点包含 m 个键码，而且如果这样，则它应以同样的方式分裂（上一小节的图 27 画出了 $m=3$ 的情况）。如果我们需要分裂根节点，根节点是没有父节点的，则只需建立包含单个键码 $K_{\lceil m/2\rceil}$的一个新根

节点即可，在这种情况下这株树长高了一级。

这个插入步骤几乎保持 B 树的所有性质；为了鉴赏这一思想的巧妙性，读者应该作习题 1。注意，这株树实质上从顶部增长，而不是从底部，因为仅当根分裂时它的高度才增加。

从 B 树删去，仅仅比插入稍稍复杂一点而已（见习题 6）。

关于性能的上限 现在让我们看看，当对阶为 m 的一株 B 树进行查找时，在最坏的情况下，要访问多少节点。假设有 N 个键码，且 $N+1$ 片叶出现在 l 级上，则在级 1,2,3,…上的节点数至少是 $2, 2\lceil m/2\rceil, 2\lceil m/2\rceil^2, \cdots$；因此

$$N+1 \geqslant 2\lceil m/2\rceil^{l-1} \tag{5}$$

换句话说，

$$l \leqslant 1 + \log_{\lceil m/2\rceil}\left(\frac{N+1}{2}\right) \tag{6}$$

这意味着，例如，如果 $N=1.999{,}998$ 且 $m=199$，则 l 至多为 3。由于我们在一次查找期间至多存取 l 个节点，故这个公式保证了运行时间十分小。

当插入一个新的键码时，可能需要分裂多达 l 个节点。然而，需要分裂的节点的平均数要小得多。因为在整株树被构造时出现的分裂总数，恰好是树中内部节点总数减 l。如果有 p 个内节点，则至少有 $1+(\lceil m/2\rceil-1)(p-1)$ 个键码；因此

$$p \leqslant 1 + \frac{N-1}{\lceil m/2\rceil - 1} \tag{7}$$

由此得出，在构造一株 N 个键码的树时，每作一次插入，我们需要分裂一个节点的次数，平均少于 $1/(\lceil m/2\rceil-1)$。

改进和变形 只要稍微突破常规，就有若干方法改进上面定义的基本 B 树结构。

首先，我们注意到，在 $l-1$ 级节点中所有指针都是 Λ，而其它级中的指针皆非 Λ。这通常表示大量空间的浪费，所以，我们可以通过消去所有的 Λ 和对所有“底”节点使用一个不同的 m 值以节省时间和空间。使用两个不同的 m 并不搅乱插入算法，因为被分裂的那个节点的两半，保留在和原来节点相同的级上。事实上我们可通过要求在 $l-k$ 级上的所有非根节点，都有 $m_k/2$ 到 m_k 个儿子，来定义阶为 $m_1, m_2, m_3, \cdots$ 的一株广义 B 树；这样一株 B 树在每级上都有不同的 m，而插入算法实质上仍和从前一样有效。

为进一步贯彻上一段中的这个思想，我们可以在树的每一级上，使用一个完全不同的节点格式，而且我们也可以在叶中存储信息。有时，键码仅构成一个文件中记录的很小部分，在这样的情况下，在靠近树根的分支节点处保存整个记录，乃是一个错误；这会使 m 对于有效的多路分支来说太小了。

我们因此可以重新考虑图 30，同时想像文件的所有记录现在都存在叶中，而且

只有少量的键码被拷贝在分支节点中。在这种解释之下，最左的叶包含其键码≤011 的所有的记录；标志为 A 的叶包含其键码满足

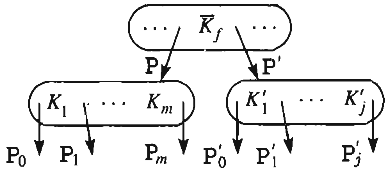
$$439 < K \leqslant 449 \tag{8}$$

的所有记录；等等。在这种解释之下，叶节点的增长和分裂就如同分支节点那样，只是一个记录决不会从一片叶传到下一级去。于是，诸叶的容量至少总是填得半满的。每当一片叶分裂时，一个新的键码便进入树的非叶部分。如果每片叶链接到它在对称次序下的后继，则我们就获得了以有效和方便的方式、既是顺序地、又是随机地来遍历文件的能力。这个变形已经作为一个 B^+ 树而闻名。

S.P.Ghosh 和 M.E.Senko[*JACM* **16** (1969), 569～579]的某些计算提示，把叶做得相当大，比如说大到大约 10 个连续的页那样长，可能是一个好的想法。通过对每片叶在已知的键码范围内的线性内插，我们可以猜测哪 10 个叶大概包含一个给定的查找变元。如果猜测是错误的，就损失了时间，但是经验指出，这个损失比我们通过减少树的大小所节省的时间要少。

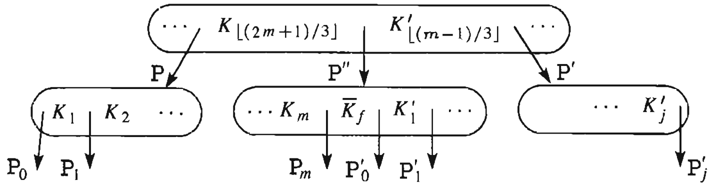
T.H.Martin[未发表]指出，奠定 B 树的基础的思想，也可以用于可变长的键码。我们不必对每个节点的儿子设置$[m/2, m]$的界限，代替的是，我们可以只是说每个节点中应该至少有半满的数据；尽管每个节点中键码的确切数目依赖于键码的长短，插入和分开的机制仍然工作得很好。然而，不应该允许键码过长，否则它们可能会把事情弄糟(见习题 5)。

对于基本 B 树方案的另一个重要的修改是 Bayer 和 McCreight 提出的溢出思想。这个思想尽可能避免频繁地分裂节点，而代之以使用一个局部转动，来改进插入算法。假设有一个节点，它由于包含 m 个键码和 $m+1$ 个指针而过满了；我们不去分裂它，而是首先在右边考察它的兄弟节点，这个兄弟节点比如说有 j 个键码和 $j+1$ 个指针。在父节点中，有一个键码 $\overline{K_f}$，它按下图分开这两个兄弟的键码



(9)

如果 $j < m-1$，则一个简单的重新排列就使分裂成为不必要的了：我们在左节点中保留$\lfloor (m+j)/2 \rfloor$个键码，在父节点中以 $K_{\lfloor (m+j)/2 \rfloor + 1}$来代替 $\overline{K_f}$，并把剩下的$\lceil (m+j)/2 \rceil$个键码(包括 $\overline{K_f}$ 在内)以及对应的指针放置到右节点中。于是这个满了的节点就"流动"到它的兄弟节点处。另一方面，如果兄弟节点已经满了($j = m-1$)，则我们可以把这两个节点都分裂为三个节点，且其中每个约装满三分之二，分别包含$\lfloor (2m-2)/3 \rfloor$、$\lfloor (2m-1)/3 \rfloor$以及$\lfloor 2m/3 \rfloor$个键码：



(10)

如果原来的节点没有右兄弟,则我们实际上可以同样的方式考虑它的左兄弟。(如果原来的节点既有右兄弟又有左兄弟,则我们甚至可以不分裂出一个新节点,除非左兄弟和右兄弟两者全都是满的)。最后,如果有待分裂的原来的节点全然没有兄弟,则它必然是根;我们可以改变 B 树的定义,允许根包含 $2\lfloor(2m-2)/3\rfloor$那么多个键码,使得当根分裂时,产生每个有$\lfloor(2m-2)/3\rfloor$个键码的两个节点。

上一段中所有技术的效果是产生一类优良的树,比如说,阶为 m 的一株 B^* 树,可以定义如下:

i) 除了根以外每个节点至多有 m 个儿子。

ii) 每个节点,除了根和叶以外,至少有 $(2m-1)/3$ 个儿子。

iii) 根至少有两个和至多有 $2\lfloor(2m-2)/3\rfloor+1$ 个儿子。

iv) 所有的叶都出现在同一级上。

v) 具有 k 个儿子的一个非叶节点,包含 $k-1$ 个键码。

重要的改动是条件(ii),它断言,我们至少利用了每个节点中三分之二的可用空间。这个改动不仅更有效地使用了空间,而且也使查找过程加快,因为在(6)和(7)中我们可以用$\lceil(2m-1)/3\rceil$代替$\lceil m/2\rceil$。然而插入过程就变慢了,因为随着节点变满它们要更加注意;关于个中涉及的折衷的分析,请参见张斌和许玖君,*Acta Informatica* **26** (1989),421~438。

就另一个极端而言,在一株频繁变动的树中使诸节点变成少于半满则更好,特别是如果插入的次数趋向于超过删去的次数时。这种情况已经由 T.Johnson 和 D.Shasha 在 *J.Comput.Syst Sci* **47** (1993),45~76 上做了分析。

由于在 B 树中根的次数可以低到 2,也许读者已经发生怀疑了:为什么我们要花费整整一次磁盘访问于仅仅一个 2-路判定上呢?! 一个简单的缓冲方案,即所谓**最近最少使用的页替换**,可以消除这种反对意见;我们可以在内存中保持若干个缓冲区的信息负载,使得当对应的叶已经出现时可以避免输入命令。在这个方案中,查找或插入算法发出"虚拟的读"命令,仅当必要的叶不在内存中时,它才被翻译成真正的输入指令;当这缓冲区已经被读过而且可能已被算法所修改时,即发出随继的"释放"命令。当需要一个真正的读入时,便选择最早释放的缓冲区;如果该缓冲区的内容在它们被读入以后发生了变化,则我们写出该缓冲区,然后就把所希望的页读入到这个选定的缓冲区中。

由于树中的级数相对于缓冲区数来说一般都较小,这个分页的方案将确保根页总在内存中;而且如果根只有 2 个或 3 个儿子,则第一级的页面几乎肯定地也将驻

留于内存中。在一次插入期间,任何可能需要被分开的页,都可以在需要时自动地出现在内存中,因为从紧挨的此前的查找会把它们记住。

E.McCreight 的某些经验表明,这个思想是十分成功的。例如,他发现,对于 10 个缓冲区和 $m=121$,以递增次序插入 100,000 个键码的过程仅需要 22 条实际的读命令和 857 条实际的写命令;于是,大多数活动都在内存中进行。而且这株树仅仅包含 835 个节点,只比极小可能的值$\lceil 100000/(m-1)\rceil=834$ 大 1;因此存储利用接近 100%。对于这个实验,他使用了溢出技术,但如同(4)中那样仅通过 2-路节点分裂,而不是像在(10)中那样的 3-路分裂(见习题 3)。

在另一个实验中,再次通过 10 个缓冲区和 $m=121$ 以及溢出技术,他以随机次序把 5000 个键码插入到初始为空的一株树中。在作了 2762 个实际的读和 2739 个实际的写之后,便产生了具有 48 个节点的一株 2 级树(87% 的存储利用)。随后的 1000 个随机查找需要 786 个实际的读。在作了 2743 个实际的读和 2800 个实际的写之后,没有溢出特征的同样实验,产生了具有 62 个节点的一株 2 级树(67% 的存储利用);1000 个相继的随机查找需要 836 次实际的读。这不仅表明分页方案是有效的,而且表明在判定分裂一个节点之前,局部地处理溢出是明智的。

姚期智已经证明,对于很大的 N 和 m,在没有溢出特征的随机插入之后的平均节点数将是 $N/(m\ln 2)+O(N/m^2)$,所以存储利用将近似为 $\ln 2\approx 69.3\%$[*Acta Informatica*, **9**(1978),159~170]。也请参见 B.Eisenbarth、N.Ziviani、G.H.Gonnet、K.Mehlhorn,以及 D.Wood,*Information and Control* **55**(1982),125~174;R.A.Baeza-Yates,*Acta Informatica* **26**(1989),439~471 所做的更详细的分析。

在 B 树被发明之后,它们很快就流行起来。例如,参见 Douglas Comer 在 *Computing Surveys* **11**(1979),121~138,412 上的文章,它讨论了早期的发展并且描述了由 IBM 公司开发的称为 VSAM(Virtual Storage Access Method,虚拟存储访问方法)的一个广泛使用的系统。VSAM 的创新之一是复制块区到磁盘磁道上以极小化等待时间。

不幸的是,对基本 B 树策略的两个最有兴趣的发展赋予了两个几乎相同的名字“*SB*-树”和“SB-树”。P.E.O'Neil[*Acta Inf.* **29**(1992),241~265]被设计来通过对邻近的记录分配相同的道或圆柱面,从而在需要同时访问许多连续的记录的应用中维持有效性,来极小化磁磁盘的输入/输出时间;在这种情况下“*SB*”以斜体表示而其中的 S 表示“顺序”。而由 P.Ferragina 和 R.Grossi 提出的 *SB*-树[*STOC* **27**(1995),693~702;*SODA* **7**(1996),373~382]是 B 树结构同我们将在 6.3 节中讨论的 Patricia 树的一个优美组合。在这种情况下“SB”是以罗马体表示而其中的 S 表示“串”。SB-树对于大型文本处理有许多应用,而且它们提供了在磁盘上对可变长的串的有效排序的基础[参见 Arge,Ferragina,Grossi 和 Vitter,*STOC* 29(1997),540~548]。

习 题

1.[*10*] 在把键码 613 插入到图 30 中之后,得到什么样的阶为 7 的 B 树?(请勿用“溢出”技术。)

2.[*15*] 做习题 1,但使用溢出技术和如同(10)中那样的 3 路分裂。

▶3.[*23*] 假如我们以递增的次序把键码 1,2,3,…插入到初始为空的阶为 101 的 B 树中,

(a) 当不使用溢出时;

(b) 当使用溢出并且如(4)那样的仅仅 2 路的分裂时;

(c) 当使用一株阶为 101 的 B^* 树、溢出以及如同(10)中的 3 路分裂时,问哪一个键码首先引起诸叶出现在级 4 上?

4.[*21*] (Bayer McCreight) 说明,如何处理对一株广义的 B 树的插入,使得除根和叶之外的所有节点都保证至少有 $\frac{3}{4}m-\frac{1}{2}$ 个儿子。

▶5.[*21*] 假设某个节点表示 1000 个字符的外存位置。如果每个指针占用 5 个字符,且键码是可变长的,其长度在 5~50 个字符之间,但总是 5 的倍数,则在一次插入期间,分裂一个节点之后,该节点中至少有多少字符位置被占用?(只考虑类似于正文中对固定长的 B 树所述的那种简单的分裂过程,而不使用“溢出”;上移使剩下的两部分最接近相等的键码。)

6.[*23*] 试设计 B 树的一个删去算法。

7.[*28*] 试设计 B 树的一个连接算法(参见 6.2.3 小节)。

▶8.[*HM37*] 考虑 Muntz 和 Uzgalis 所建议的树插入的推广,其中每页可保持 M 个键码。在把 N 个随机项目插入到这样一株树中,使得它有 $M+1$ 个外节点之后,设 $b_{Nk}^{(j)}$ 是具有如下性质的一次不成功查找的概率,该查找需要 k 次页访问,并结束于一个外节点,该节点的父节点属于一个包含 j 个键码的页,如果 $B_N^{(j)}(z)=\sum b_{Nk}^{(j)}z^k$ 是对应的生成函数,试证明,我们有 $B_1^{(j)}(z)=\delta_{j1}z$,而且

$$B_N^{(j)}(z)=\frac{N-j-1}{N+1}B_{N-1}^{(j)}(z)+\frac{j+1}{N+1}B_{N-1}^{(j-1)}(z),\ \text{对于 } 1<j<M$$

$$B_N^{(1)}(z)=\frac{N-2}{N+1}B_{N-1}^{(1)}(z)+\frac{2z}{N+1}B_{N-1}^{(M)}(z)$$

$$B_N^{(M)}(z)=\frac{N-1}{N+1}B_{N-1}^{(M)}(z)+\frac{M+1}{N+1}B_{N-1}^{(M-1)}(z)$$

试求每次不成功查找的平均页访问次数 $C'_N=\sum_{j=1}^{M}B_N^{(j)\prime}(1)$的渐近特性。[提示:借助矩阵

$$W(z)=\begin{pmatrix}-3 & 0 & \cdots & 0 & 2z\\ 3 & -4 & \cdots & 0 & 0\\ 0 & 4 & \cdots & 0 & 0\\ \vdots & \vdots & & \vdots & \vdots\\ 0 & 0 & \cdots & -M-1 & 0\\ 0 & 0 & \cdots & M+1 & -2\end{pmatrix}$$

来表达这个递推式,并把 C'_N 同 $W(1)$中的 N 次多项式联系起来。]

9.[*22*] B 树的思想能否用来通过位置而不是通过键码值来查找一个线性表的项?(参见算法 6.2.3B)。

10.[35] 试讨论如何能把组织成一个 B 树的大型文件以下列这样一种方式，为大量同时的用户用作并发访问和修改，使不同页的用户彼此很少相干扰。

Little is known, even for otherwise equivalent algorithms,
about the optimization of storage allocation,
minimization of the number of required operations,
and so on. This area of investigation
must draw upon the most powerful resources
of both pure and applied mathematics
for further progress.
关于存储分配的优化，关于所需要的操作个数的极小化，等等，
撇开这些即使是对于就等价的算法，我们知之甚少。为了取得更大进步，
这个研究领域必须投入纯数学和应用数学两方面最强有力的资源。
—ANTHONY G. OETTINGER (1961)

6.3　数字查找

代替把一个查找方法建立在键码之间比较的基础上，我们可以利用它们的数字序列或字母字符序列的表示。例如，考虑一部大型字典中的"书边标目"；由一个给定的字的第一个字母，我们可以立即定出包含所有以该字母开始的字的那些页的位置。

如果我们推广书边标目的思想，则它的逻辑结论之一就是如表 1 所说明的以重复"下标"为基础的一个查找方案。假设我们要检测一个给定的查找变元，看它是否是英语中最普通的 31 个字之一(参见 6.2.2 小节中的图 12 和 13)，这个数据在表 1 中被表示成所谓的"trie"(检索)结构，这个名称是由 E.Fredkin[*CACM* **3** (1960), 490～500]提议的，因为它是信息检索(retrieval)的一部分。一个检索结构实际上是一株 M 叉树，它的节点是 M 维向量，其分量对应于数字或字符，第 l 级上的每个节点表示所有这样的键码的集合，它们以 l 个字符的某个序列开始，这个序列称做键码的前缀；这个节点根据第 $l+1$ 个字符确定一个 M 路分支。

例如，表 1 的检索结构有 12 个节点；节点(1)是根，我们在这里查找头一个字符。如果第一个字符是 N，则这张表告诉我们，我们的字必然是 NOT(否则它不在表中)。另一方面，如果第一个字母是 W，则节点(1)告诉我们转到节点(9)；以同一方式查找第二个字母，节点(9)指出第二个字母应该是 A、H 或 I。节点(10)的前缀是 HA。空白条款代表空的链接。

表 1 中节点向量是按照 MIX 字符编码重新排列过的。这意味着一个检索结构的查找将是十分快的，因为我们仅仅通过使用我们的键码的字符作为下标，来取出一个阵列中的字。通过下标进行快速多路判断的技术，称为"检表"(Table Look-At)，以示区别于"查表"(Table Look-Up)[见 P.M.Sherman, *CACM* **4**(1961), 172～173, 175]。

表 1 31个最普遍的英文字的一个检索结构

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)
␣		A				I					HE	
A	(2)				(10)				WAS			THE
B	(3)											
C												
D										HAD		
E			BE		(11)							THE
F	(4)						OF					
G												
H	(5)							(12)	WHICH			
I	(6)				HIS				WITH			THIS
Δ												
J												
K												
L												
M												
N	NOT	AND				IN	ON					
O	(7)			FOR				TO				
P												
Q												
R		ARE		FROM			OR				HER	
Σ												
Π												
S		AS				IS						
T	(8)	AT				IT						
U			BUT									
V										HAVE		
W	(9)											
X												
Y	YOU		BY									
Z												

算法 T(检索结构查找) 给定一个形成 M 叉检索结构的记录表,本算法查找一个给定的变元 K。这个检索结构的节点,都是其下标从 0 变到 $M-1$ 的向量;这些向量的每一分量或者是一个键码或者是一个链接(可能是空的)。

T1. [初始化] 置链接变量 P,使得它指向检索结构的根。

T2. [分支] 置 k 为输入变元 K 的从左到右的下一个字符(如果这个变元已经完全扫描过了,则置 k 为一个“空白”或字结束符号。字符应该表示为在范围 $0 \leqslant k < M$ 中的数)。设 X 为 NODE(P)中编号为 k 的表项目。如果 X 是一个链接,则转到 T3;但如果 X 是一个键码,则转到 T4。

T3. [前进] 如果 $X \neq \Lambda$,则置 P←X 并返回步骤 T2;否则算法以失败告终。

T4. [比较] 如果 $X = K$,则算法成功地结束,否则它以失败告终。 ▎

注意,如果这个查找是不成功的,则已经找到了最长的匹配。这个性质在应用上有时是有用的。

为了比较这个算法和这一章中其它算法的速度,我们可以写出一个短的 MIX 程序,同时假定字符都是字节而且键码的长度至多是五个字节。

程序 T(检索结构的查找) 这个程序假定所有键码都表示成一个 MIX 字,当键码少于五个字符时,空白字符在右边。由于我们使用 MIX 字符代码,因此假定查找变元的每个字节都包含小于 30 的一个数。链接被表示作一个节点字的 0:2 字段中的一个负数。rI1≡P,rX≡K 的未扫描部分。

```
START    LDX    K           1    T1. 初始化
         ENT1   ROOT        1    P←指向检索结构的根的指针
2H       SLAX   1           C    T2. 分支
         STA    *+1(2:2)    C    摘出下一个字符 k
         ENT2   0,1         C    Q←P+k
         LD1N   0,2(0:2)    C    P←LINK(Q)
         J1P    2B          C    T3. 前进。如果 P 是一个≠Λ
                                 的链接则转到 T2
         LDA    0,2         1    T4. 比较。rA←KEY(Q)
         CMPA   K           1
         JE     SUCCESS     1    如果 rA=K,则成功地转出
FAILURE  EQU    *                如果不在检索结构中则转出 ▌
```

这个程序的运行时间是 $8C+8$ 个单位,其中 C 是考察的字符数。由于 $C\leqslant 5$,故这个查找决不花费多于 48 个时间单位。

如果我们现在要比较这个程序(使用表 1 的检索结构)同程序 6.2.2T(使用图 13 的最优二分查找树)的效率,则可以作如下的观察。

1. 检索结构花费更多得多的内存空间;仅仅为了表示 31 个键码,我们就使用了 360 个字,而二分查找树只使用 62 个内存字。(然而,习题 4 表明,通过省去某些无用的位置,实际上可以把表 1 的检索结构放在仅仅 49 个字中。)

2. 对两个程序来说,每次成功的查找都花费 26 个时间单位。但是一次不成功的查找在检索结构中将进行得更快些,而在二分查找树中则要慢些。对于这个数据来说查找的不成功次数多于成功的次数,所以从速度观点看,检索结构是更可取的。

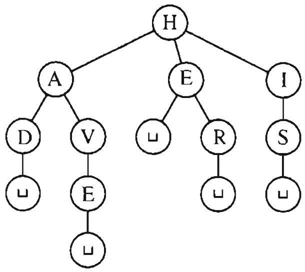
3. 如果我们应用的对象是图 15 的 KWIC 索引而不是 31 个最普通的英文字,则由于这个数据的特性,检索结构就失去它的优越性了。例如,一个检索结构为了区别 COMPUTATION 和 COMPUTATIONS 竟需要作 12 次迭代。这种情况下,若把检索结构构造成从右到左扫描字,而不是从左到右,则更好些。

表示一族串的检索结构的抽象概念是由 Axel Thue 在不包含相邻重复子串的一篇论文[*Skrifter udgivne af Videnskabs-Selskabet i Christiania, Mathematisk-*Naturvidenskabelig Klasse(1912)No.1,重新印刷在 Thue 的 *Selected Mathematical Papers* (Oslo: Universitetsforlaget, 1977), 413~477]中引进的。

用于计算机查找的检索结构的存储的思想,首先由 Rene. de. la Braindais 推荐的[*Proc. Western Joint Computer Conf.* **15**(1959), 295~298]。他指出,如果我们对每个节点向量都使用一个链接表,则可以以运行时间为代价来节省内存空间,因为大

多数向量的元素趋向于空。实质上，这个思想在于通过图 31 中所示的森林来代替表 1 的检索结构。在这样一个森林中，查找是通过如下方式进行的：首先找到与第一个字符匹配的根，然后找到与第二个字符匹配的该根的子根，等等。

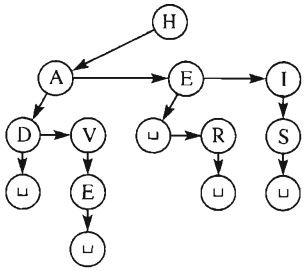
在这篇论文中，de la Briandais 并不完全像表 1 或图 31 中所示的那样停止树的分叉；而是继续一个字符挨一个字符地表示每个键码，直到达到字结束的限定符为止。于是，他实际上使用了



(1)

来代替图 31 中的“H”树。这个表示要求更多的存储，但是它使得可变长数据的处理特别容易。如果我们对每个字符使用两个链接字段，则动态插入和删去即可以用一种简单的方式来处理。

如果我们使用通常的方法把树表示作二叉树，则(1)就变成二叉树(在图 31 的完全森林表示中，我们也应有一个从 H 向右引到其邻近根 I 的一个指针)。在这株二叉树中，查找如下进行：把变元中的一个字符同树中的字符加以比较，沿着诸 RLINK 直到找到一个匹配为止；然后取 LLINK 并以同一方式来处理变元的下一个字符。



(2)

对于这样一株二叉树，我们或多或少地通过比较法进行查找，用相等-不相等的分支代替了小于-大于的分支。6.2.1 小节的基础理论告诉我们，为了区别 N 个键码，平均说来，必须至少作 $\lg N$ 次比较；当查找像图 31 那样的一株树时，所作的平均查找次数，至少必须同使用 6.2 节的技术进行一次二分查找时所作的平均检验次数一般多。

另一方面，表 1 中的检索结构一次就能进行整个 M 路分支；我们将看到，如果

输入数据是随机的,则对于很大的 N 的平均查找时间,大约只要

$$\log_M N = \lg N/\lg M$$

次迭代。我们还将看到,类似算法 T 中的一个"纯粹"检索结构方案,需要总数约 $N/\ln N$ 个节点对 N 个随机输入进行区别;因此总共的空间数量同 $MN/\ln M$ 成正比。

从这些考虑显然可看出,检索结构的思想仅在树的头几级中有益。把两种策略混合使用,即对头一些字符使用检索结构,而后又转到某种其它的技术上,我们可以获得更好的性能。例如,E.H.Sussenguth,Jr[*CACM* **6**(1963),272~279]曾提议把逐个字符的方案一直用到树的那样的部分,其中,比如说,文件中的键码至多是六个,然后,我们就能顺序地扫视剩下的键码的短表。我们将看到,这个混合策略能减少约六分之一的检索结构节点数,而运行时间并没有很大的变化。S.Y.Berkovich 在*Doklady Akademii Nauk SSSR* **202**(1972),298~299[英译:*Soviet Physics-Doklady* **17** (1972),20~21]中提出一个有趣的方法,该方法用于存储在外存中不断增长的大的检索结构。

T.N.Turba[*CACM* **25**(1982),522~526]指出,有时,通过对于每个不同的长度都有一株查找树或检索结构,来对可变长的键码进行查找,是最方便的。

二进的情形 现在考虑 $M=2$ 的特殊情况,在这种情况下,我们每次扫描查找变元的一个二进位。已经提出了两种有趣的方法,它们特别适合于这一情况。

第一种方法,我们将称之为数字树查找,它是由 E.G.Coffman 和 J.Eve[*CACM* **13**(1970),427~432,436]给出的。这个思想同 6.2.2 小节树查找算法中所做的完全一样,在该节点中存储全键码,但使用该变元的二进位(而不是用比较的结果)来支配在每步中是取左分支还是取右分支。图 32 表示,当我们以递减频率的次序插入 31 个最常用英文字时,用这种方法所构成的树。为了给这个图提供二进数据,先把这些字表达成 MIX 的字符代码,然后再把字符代码转换成每个字节 5 个二进位的二进数。例如,字 WHICH 表示为"11010 01000 01001 00011 01000"。

为了在图 32 中查找 WHICH 这个字,我们首先把它同该树根处的 THE 进行比较。因为没有匹配,且 WHICH 的第一个二进位是 1,我们向右移而同 OF 作比较。因为还不匹配,且因 WHICH 的第二个二进位是 1,我们向右移,而同 WITH 进行比较,等等。在一个数字查找树中键码的字母顺序不再对应于节点的对称顺序。

把图 32 和 6.2.2 小节中的图 12 作一对照是有趣的,因为后一株树是以同一方式形成的,但是用比较而不是用用于转移的键码二进位形成的。如果我们考虑给定的频率,则图 32 的数字查找树要求对每次成功的查找平均进行 3.42 次比较;这比图 12 所需要的 4.04 次比较稍微好些,尽管每次比较花费的时间可能不同。

算法 D(数字树查找) 给定一个记录表,它形成如上所述的一株二叉树。本算法查找给定的变元 K。如果 K 不在表中,则把包含有 K 的新节点插入这株树的适当位置中。

本算法假定,这株树是非空的且其节点如算法 6.2.2T 中那样,有 KEY、LLINK 和 RLINK 字段。事实上,读者可以验证,这两个算法几乎是相同的。

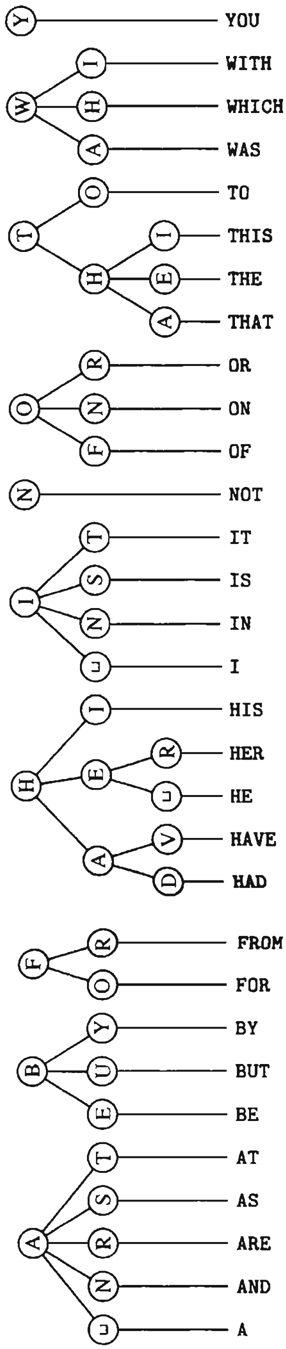


图 31　表 1 的检索结构转换为一片森林

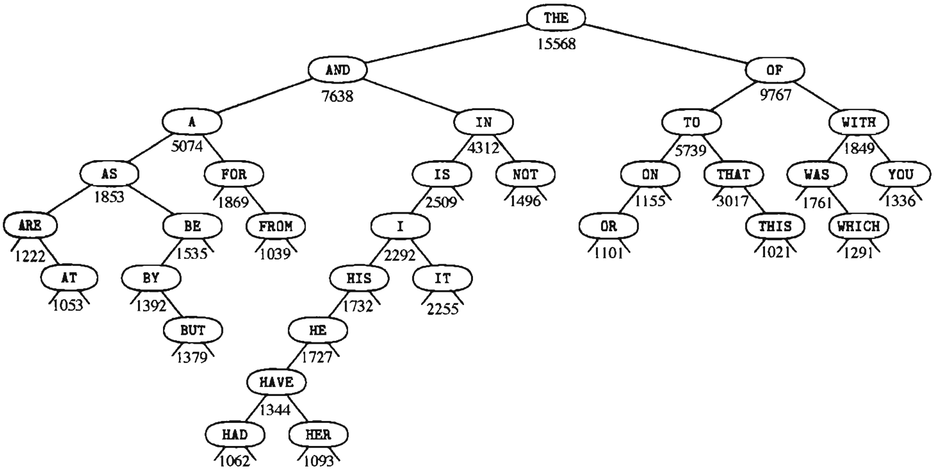


图 32　对于以递减频率次序插入的 31 个最常用英语单字的一株数字查找树

D1.[初始化] 置 P←ROOT,且 K'←K。

D2. [比较] 如果 K = KEY(P),则这个查找成功地结束。否则置 b 为 K' 的第一个二进位,并且把 K' 左移一位(由此删去该二进位并在右边引入一个 0)。如果 $b=0$,则转到 D3,否则转到 D4。

D3. [左移] 如果 LLINK(P)≠Λ,则置 P←LLINK(P)并转回到 D2,否则转到 D5。

D4.[右移] 如果 RLINK(P)≠Λ,则置 P←RLINK(P)并转回到 D2。

D5. [插入树中] 置 Q⇐AVAIL,KEY(Q)←K,LLINK(Q)←RLINK(Q)←Λ。如果 $b=0$ 则置 LLINK(P)←Q,否则置 RLINK(P)←Q。 ▌

尽管算法 6.2.2T 的树查找本质上是二进的,但不难看出,现在的算法可以被扩充成对任何 $M\geqslant 2$ 的一个 M 叉数字查找(见习题 13)。

Donald R. Morrison[*JACM* **15**(1968),514～534]已经发现一个非常好的方法形成以键码的二进表示为基础的 N 节点查找树,而无需在节点中存储键码。他的方法,称为帕特里西亚(“Patricia”)(是 Practical Algorithm To Retrieve Information Coded In Alphanumeric 字头拼写成的)①,特别适合于处理极其长的、可变长的键码,例如存储在一个大型文件中的标题或短句。

Patricia 的基本思想是要构造一个二进的检索结构,但在每个节点中都包含了在作下一个检验之前可以跳过的二进位的位数,以避免单路分支。有若干方法来说明这一思想;也许最便于说明的如图 33 中所示。我们有一个二进位的 TEXT 阵列,它通常都十分冗长;它可以作为一个外部直接存取文件存储,因为每次查找仅访问 TEXT 一次。有待存入我们表中的每个键码,都由文本中的一个开始位置确定,并且

① 意指检索按字母数字编码的信息的实用算法。——译者注

总可把它想像成要由这个开始位置起直至达到文本的末尾为止(Patricia 不查找键码和变元之间的严格相等性,而是来确定是否存在以该变元开始的一个键)。

图 33 中所描述的情况包含 7 个键码,在每个字处都开始一个键码,即“THIS IS THE HOUSE THAT JACK BUILT?”(这是杰克建造的房子?)以及“IS THE HOUSE THAT JACK BUILT?”(是杰克建筑的房子吗?)以及……以及“BUILT?”(建造?)。有一个重要的限制,即没有任何一个键码可以是另一个键码的前缀;如果我们以一个在别处都不出现的惟一的文本结束代码(在现在情况下是“?”),来结束文本,则这个限制即可满足。同样的限制也隐含在算法 T 的检索结构中,在那里“␣”是结束代码。

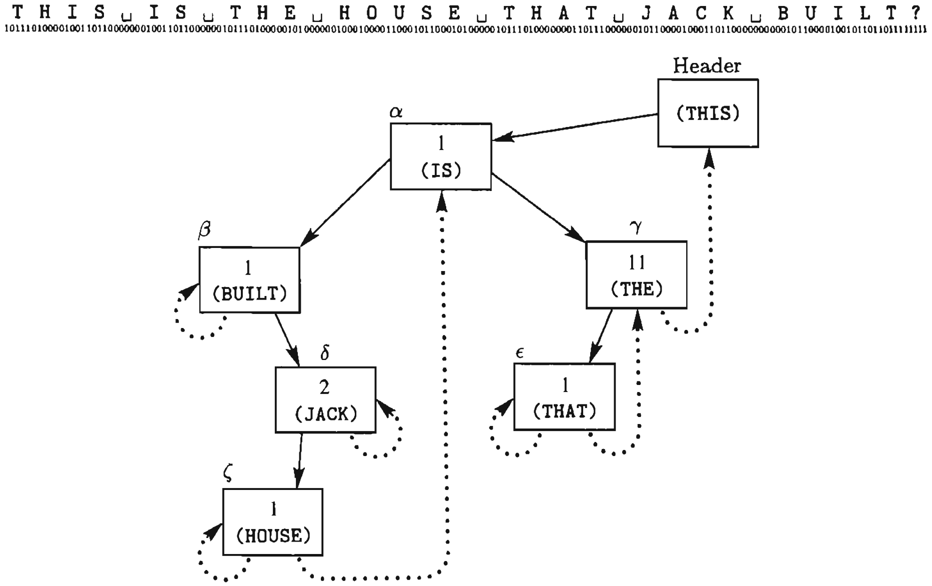


图 33 Patricia 树和 TEXT 的一个示例

Patricia 用以查找的这株树,应当包含在随机存取存储器中,或被安排在 6.2.4 小节所建议的一些页上。它由一个表头和 $N-1$ 个节点所组成,这些节点都包含若干个字段:

KEY,指向文本的一个指针。如果文本含 C 个字符,则这个字段的长度必须至少有 $\lg C$ 个二进位。在图 33 中,每个节点内所示的字,实际上都可由指向文本的指针来表示;例如,该节点不是包含“(JACK)”,而将包含数 24(它指出在文本串中“JACK BUILT”的开始位置)。

LLINK 和 RLINK,在树内的指针。这些字段的长度至少必须是 $\lg N$ 个二进位。

LTAG 和 RTAG,两个单二进位的字段,它们分别表示 LLINK 和 RLINK 是指向这个节点的儿子还是祖宗的指针。图 33 中的虚线对应于其 TAG 二进位为 1 的指针。

SKIP,如下所述,它是一个数,说明当查找时应跳过多少二进位。这个字段应该足够大,使得对于作为至少两个不同键码的一个前缀的某个串 σ,具有前缀 σ 的所有键码在紧接着 σ 之后的下 k 个二进位相一致。实际上,我们通常可以假定 k 不太大,如果它超过了 SKIP 字段的大小,则可给出一个错误指示。SKIP 字段在图 33 中表示为每个节点内的数。

表头仅含 KEY、LLINK 和 LTAG 字段。

在 Patricia 树中的查找是如下进行的:假设我们正在查找字 THE(二进位的型式为 10111 01000 00101)。我们从寻找根节点 α 的 SKIP 字段开始,它告诉我们检查这个变元的头一个二进位。该位是 1,所以向右移。下一个节点 γ 的 SKIP 字段告诉我们寻找这个变元的第 $1+11=12$ 位。该位为 0,所以左移。下一个节点 ϵ 的 SKIP 字段告诉我们寻找第$(12+1)$位,该位是 1;现在找到 RIAG = 1,所以回到节点 γ,它让我们参考 TEXT。我们所走过的查找通路将对其二进位型式是 1×××× ×××××× 01…的任何变元出现,因此必须检验,看看它是否匹配以该型式开始的惟一键码即 THE。

另一方面,假设我们寻找一个或所有以 TH 开始的键码。这个查找过程同上面所述的过程一样开始,但它最后试图查找 10 位变元的(不存在的)第 12 位。这时,我们在当前节点所确定的点处(在现在情况下为节点 γ),把变元同 TEXT 作比较。如果它不匹配,则此变元就不是任何键码的开头;但如果它匹配了,则此变元就是每一个由节点 γ 和其后裔中的虚线所表示的键码(即 THIS,THAT,THE)的开头。

这个过程可以更精确地叙述如下。

算法 P(Patricia) 给定一个 TEXT 阵列和一株具有如上所述 KEY、LLINK、RLINK、LTAG、RTAG 以及 SKIP 字段的树,本算法确定在这个 TEXT 中是否有一个以特定变元 K 开始的键码。(如果对于 $r\geqslant 1$,存在 r 个这样的键码,则随后有可能在 $O(r)$ 步内来确定所有它们的位置;见习题 14。)我们假定至少存在一个这样的键码。

P1.[初始化] 置 P←HEAD 和 $j\leftarrow 0$。(变量 P 是一个沿此树下移的指针,而 j 是一个计数器,它将标记变元的二进位的位置)。置 $n\leftarrow K$ 中二进位的个数。

P2.[左移] 置 Q←P 和 P←LLINK(Q)。如果 LTAG(Q) = 1,则转到 P6。

P3.[跳过二进位](这时我们知道,如果 K 的前 j 个二进位同无论哪一个键码匹配,则它们都同在 KEY(P)处开始的键码匹配。)置 $j\leftarrow j+$ SKIP(P)。如果 $j>n$,转到 P6。

P4.[检验二进位](这时我们知道,如果 K 的前 $j-1$ 个二进位同无论哪一个键码匹配,它们都同在 KEY(P)处开始的键码匹配。)如果 K 的第 j 个二进位为 0,则转到 P2,否则转到 P5。

P5.[右移] 置 Q←P 和 P←RLINK(Q)。如果 RTAG(Q) = 0,则转到 P3。

P6.[比较](这时我们知道,如果 K 同任何一个键码匹配,它都同在 KEY(P)处开始的键码匹配。)K 同 TEXT 阵列中位于 KEY(P)处开始的键码作比较。如果它们相等(直到 n 位,即 K 的长度为止),则这个算法成功地结束;如果

不相等，则它以失败告终。 ▌

习题 15 说明了首先可以怎样来构造 Patricia 树。我们也可以把新的内容加到文本中和插入新的键码，只要新的文本材料总是以惟一的限定符（例如，一个文本结束符号后面接一个序列号）结尾即可。

Patricia 有一点技巧，而只有仔细地阅读，才能揭示它的所有美妙之处。

算法的分析　在结束本节之前，让我们对检索结构、数字查找树以及 Patricia 进行一番数学研究。这些分析的主要结果在最后综述。

我们首先考虑二叉检索结构的情况，即 $M=2$ 的检索结构的情况。图 34 表示，当第 5 章排序例子的十六个键码被处理作 10 位的二进数时，所形成的二进检索结构［这些键码均以八进制记法示出，例如 1144 表示 10 位数 $612=(1001100100)_2$］。如同在算法 T 中那样，我们使用检索结构来存储键码的前导二进位的信息，直到达到了键码被惟一确定的第一个位置为止；然后此键码被全部重新记录。

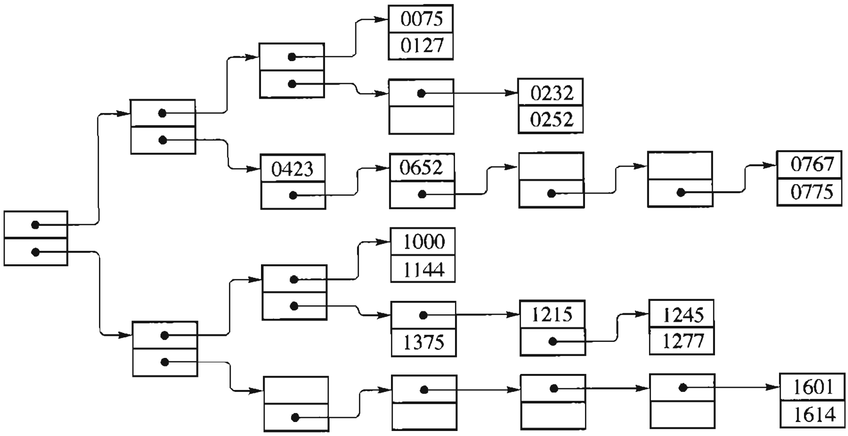


图 34　一个随机二叉检索结构的例子

如果把图 34 同表 5.2.2-3 进行比较，便可显示出检索结构存储同基数交换排序之间的令人惊异的关系（再又，这个关系也许是很显然的）。图 34 的 22 个节点精确地对应于表 5.2.2-3 中的 22 个分划阶段，而且在前根次序下的第 p 个节点对应于阶段 p。在一个分划阶段中二进位的检索数，等于在对应的节点和其子检索结构内的键码的个数；因此，我们可以叙述下列结果。

定理 T　*如果按如上所述把 N 个不同的二进数放置到一个二进检索结构中，则（i）这个检索结构的节点数就等于把这些数按基数交换排序所需要的分划阶段数；以及（ii）借助于算法 T 检索一个键码所需要的二进位探查的平均次数，等于 $1/N$ 乘以按交换排序所需要的二进位探查数。* ▌

由于这个定理，我们可以利用在 5.2.2 小节中为进行基数交换而研制的全部数

学机器。例如，如果假定，我们的键码是在 0～1 之间一致分布的无限精度的随机实数，那么，为进行检索所需要的二进位探查数将是 $\lg N+\gamma/\ln 2+1/2+\delta(N)+O(N^{-1})$，而检索结构的节点数将是 $N/\ln 2+N\bar{\delta}(N)+O(1)$。这里，$\delta(N)$和$\bar{\delta}(N)$是可以忽略的复杂函数，因为它们的值总小于 10^{-6}（见习题 5.2.2-38 和 5.2.2-48）。

当然，还有更多的工作犹待完成，因为需要把二进检索结构推广到 M 进检索结构。我们在这里只描述研究的起点，而把有益的细节留作习题。

设 A_N 是在包含 N 个键码的随机 M 进检索结构中的平均内节点数。于是 $A_0=A_1=0$，且对于 $N\geqslant 2$，我们有

$$A_N = 1+\sum_{k_1+\cdots+k_M=N}\left(\frac{N!}{k_1!\cdots k_M!}M^{-N}\right)(A_{k_1}+\cdots+A_{k_M}) \tag{3}$$

因为 $N!\ M^{-N}/k!\ \cdots k_M!$ 是 k_1 个键码在第一个子检索结构中，…，k_M 个键码在第 M 个子检索结构中的概率。利用对称性然后对 $k_2,\cdots,k_M$ 求和，这个等式可改写成

$$A_N = 1+M^{1-N}\sum_{k_1+\cdots+k_M=N}\left(\frac{N!}{k_1!\cdots k_M!}\right)A_{k_1} =$$

$$1+M^{1-N}\sum_k\binom{N}{k}(M-1)^{N-k}A_k \quad 对于\ N\geqslant 2 \tag{4}$$

类似地，如果 C_N 表示为找出检索结构中所有 N 个键码所需要的二进位探查的平均总和数，则我们发现 $C_0=C_1=0$ 且

$$C_N = N+M^{1-N}\sum_k\binom{N}{k}(M-1)^{N-k}C_k \quad 对于\ N\geqslant 2 \tag{5}$$

习题 17 说明了怎样处理这种类型的一般递推式，而习题 18-25 给出了对应的随机检索结构的理论[对于 A_N 的分析，首先是由 L.R.Johnson 和 M.H.McAndrew 联系到一个等价的面向硬件的排序算法时从另一种观点解决的，见*IBM J.Res. and Devel.* **8**(1964)，189～193]。

如果我们现在回到数字查找树的研究上，则会发现这些公式都是类似的，但又有足够的不同，即是，不容易看出怎样导出渐近特性来。例如，如果 $\overline{C}_N$ 表示当在一株二叉数字查找树中，找出全部 N 个键码时所作的二进位探查的平均总数，则和上面一样，不难导出 $\overline{C}_0=\overline{C}_1=0$，且

$$\overline{C}_{N+1} = N+M^{1-N}\sum_k\binom{N}{k}(M-1)^{N-k}\overline{C}_k \quad 对于\ N\geqslant 0 \tag{6}$$

这几乎和等式(5)相同；但是在这个等式左边出现 $N+1$ 而不是 N，这足以改变递推式的整个特征，因此我们用来研究(5)的诸方法都不能用了。

现在让我们首先考虑二进制的情况。图 35 示出，当已按第 5 章的例子所用的次序插入 16 个示例性的键码时，对应于图 34 的数字查找树。如果我们要确定在一次成功的随机查找中所作的二进位探查平均次数，则它恰巧是这株树的内部路径长

度除以 N,因为我们需要 l 个二进的探查来找出级 l 上的一个节点。然而,要注意的是,在一次不成功的随机查找中所作的二进位探查的平均次数,并非简单地同这株树的外部路径长度相关,因为不成功的查找更可能出现在靠近根的外部节点处;于是,达到图 35 中节点 0075 的左子分支的概率是 $\frac{1}{8}$(假定是无限精确度的键码),而遇到节点 0232 的左子分支的概率仅为 $\frac{1}{32}$。由于这个原因,当诸键码一致分布时,数字查找树一般来说比算法 6.2.2T 的二分查找树能更好地维持平衡。

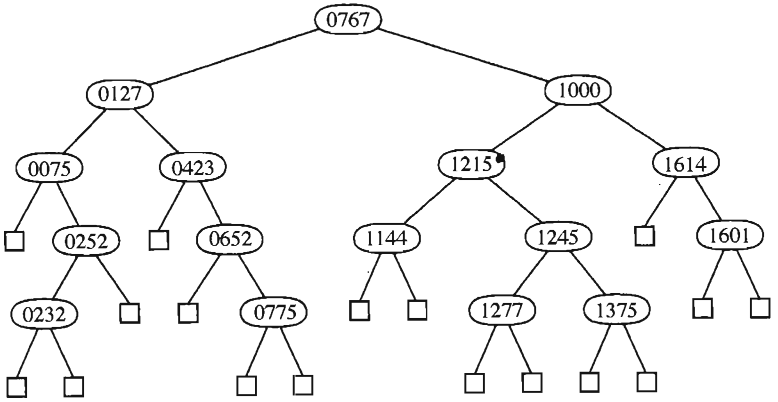


图 35 由算法 D 构造的一株随机数字查找树

我们可以用生产函数描写一个数字查找树的有关特征。如果在级 l 上有 a_l 个内节点,则考虑生成函数 $a(z)=\sum_l a_l z^l$;例如,对应于图 35 的生成函数为 $a(z)=1+2z+4z^2+5z^3+4z^4$。如果在级 l 上有 b_l 个外节点,且如果 $b(z)=\sum_l b_l z^l$,则由习题 6.2.1-25 有

$$b(z) = 1 + (2z - 1)a(z) \tag{7}$$

例如,$1+(2z-1)(1+2z+4z^2+5z^3+4z^4)=3z^3+6z^4+8z^5$。在一次随机的成功查找中,所作的二进位探查的平均次数是 $a'(1)/a(1)$,因为 $a'(1)$是树的内部路径长度,而 $a(1)$是内节点数。在一次随机的不成功的查找中,所作的二进位探查的平均次数是$\sum_l lb_l 2^{-l}=\frac{1}{2}b'\left(\frac{1}{2}\right)=a\left(\frac{1}{2}\right)$,因为我们是以 2^{-l}的概率在级 l 上的一个给定外节点处结束的。比较次数等同于二进位探查的次数,在一次成功的查找中此数要加 1。例如,在图 35 中,平均来说,一次成功的查找将花费 $2\frac{9}{16}$次二进位探查和 $3\frac{9}{16}$次比较;一次不成功查找的二进位探查和比较次数都将是 $3\frac{7}{8}$。

现在设 $g_N(z)$是具有 N 个节点的树的"平均"$a(z)$;换言之,$g_N(z)$是对于具有 N 个内节点的所有二进数字查找树的求和式$\sum p_T a_T(z)$,其中 $a_T(z)$是 T 的内节

点的生成函数，p_r 是当使用算法 D 插入 N 个随机数时 T 出现的概率。那么，在一次成功的查找中二进位探查的平均次数将是 $g'_N(1)/N$，在一次不成功的查找中为 $g_N\left(\frac{1}{2}\right)$。

我们可以模拟树的构造过程计算出 $g_N(z)$ 如下。如果 $a(z)$ 是有 N 个节点的一株树的生成函数，则我们可以通过在任何一个外节点位置上作下一次插入，而形成 $N+1$ 株树，这个插入以 2^{-l} 的概率进到级 l 上一个给定的外节点；因此，$N+1$ 株新树的生成函数乘以出现的概率再求和为 $a(z)+b\left(\frac{1}{2}z\right)=a(z)+1+(z-1)a\left(\frac{1}{2}z\right)$。对于所有具有 N 个节点的树取平均值，就得

$$g_{N+1}(z) = g_N(z) + 1 + (z-1)g_N\left(\frac{1}{2}z\right);\ g_0(z) = 0 \tag{8}$$

外节点对应的生成函数 $h_N(z)=1+(2z-1)g_N(z)$，是更为容易给出的，因为(8)等价于公式

$$h_{N+1}(z) = h_N(z) + (2z-1)h_N\left(\frac{1}{2}z\right);\ h_0(z) = 1 \tag{9}$$

重复地应用这个规则，我们求得

$$\begin{aligned} h_{N+1}(z) &= h_{N-1}(z) + 2(2z-1)h_{N-1}\left(\frac{1}{2}z\right) + (2z-1)(z-1)h_{N-1}\left(\frac{1}{4}z\right) = \\ &h_{N-2}(z) + 3(2z-1)h_{N-2}\left(\frac{1}{2}z\right) + 3(2z-1)(z-1)h_{N-2}\left(\frac{1}{4}z\right) + \\ &(2z-1)(z-1)\left(\frac{1}{2}z-1\right)h_{N-2}\left(\frac{1}{8}z\right) \end{aligned}$$

等等，于是最后我们有

$$h_N(z) = \sum_k \binom{N}{k} \prod_{j=0}^{k-1} (2^{1-j}z - 1) \tag{10}$$

$$g_N(z) = \sum_{k\geq 0} \binom{N}{k+1} \prod_{j=0}^{k-1} (2^{-j}z - 1) \tag{11}$$

例如，$g_4(z)=4+6(z-1)+4(z-1)\left(\frac{1}{2}z-1\right)+(z-1)\left(\frac{1}{2}z-1\right)\left(\frac{1}{4}z-1\right)$。这些公式使得有可能把我们正在寻找的量表达为乘积的和：

$$\overline{C}_N = g'_N(1) = \sum_{k\geq 0} \binom{N}{k+2} \prod_{j=1}^{k} (2^{-j} - 1) \tag{12}$$

$$g_N\left(\frac{1}{2}\right) = \sum_{k\geq 0} \binom{N}{k+1} \prod_{j=1}^{k} (2^{-j} - 1) = \overline{C}_{N+1} - \overline{C}_N \tag{13}$$

这个 $\overline{C}_N$ 的公式满足(6)，这全然不是显然的！

可惜，由于 $2^{-j}-1$ 是负的，这些表达式不便于计算或求出一个渐近展开式；我

们得到相当大的项并有大量的消去。应用习题 5.1.1-16 的分划恒等式，可以得到 $\overline{C}_N$ 的一个更有用的公式。我们有

$$\overline{C}_N = \Big(\prod_{j\geqslant 1}(1-2^{-j})\Big)\sum_{k\geqslant 0}\binom{N}{k+2}(-1)^k\prod_{l\geqslant 0}(1-2^{-l-k-1})^{-1} =$$

$$\Big(\prod_{j\geqslant 1}(1-2^{-j})\Big)\sum_{k\geqslant 0}\binom{N}{k+2}(-1)^k\sum_{m\geqslant 0}(2^{-k-1})^m\prod_{r=1}^{m}(1-2^{-r})^{-1} =$$

$$\sum_{m\geqslant 0}2^m\Big[\sum_k\binom{N}{k}(-2^{-m})^k-1+2^{-m}N\Big]\prod_{j\geqslant 0}(1-2^{-j-m-1}) =$$

$$\sum_{m\geqslant 0}2^m((1-2^{-m})^N-1+2^{-m}N)\sum_{n\geqslant 0}(-2^{-m-1})^n\frac{2^{-n(n-1)/2}}{\prod_{r=1}^{n}(1-2^{-r})} \quad (14)$$

乍一看去，这似乎并不是对于等式(12)的一个改进，但它有个很大的优点，即它对于每个固定的 n 对 m 的求和皆迅速收敛。对于等式 5.2.2-(38)、和 5.2.2-(39)中检索结构的情况，出现一种精确地类似的状态；事实上，如果我们仅仅考虑(14)中 $n=0$ 的项，则我们恰有 $N-1$ 加上在一个二进检索结构中的二进位探查数。现在可以继续用同以前完全一样的方式，得到这个渐近值；见习题 27。[上面的推导在很大程度上是以 A.J.Konheim 和 D.J.Newman 所提议的方法为基础的，见 *Discrete Mathematics* **4**(1973)，57～63]。

最后让我们对 Patricia 进行一点数学考察。在这种情况下，二叉树就像是在相同的一些键码上的对应的二进检索结构，但是被挤压在一起了(因为 SKIP 字段消去了 1 路分支)，因此总有 $N-1$ 个内节点和 N 个外节点。图 36 示出了对应于图 34 的检索结构中 16 个键码的 Patricia 树。在每个分支节点中所示的数是 SKIP 的数量；尽管外节点并不明显地出现，但键码还是用外节点来标记(实际上每个外节点都用一个带标志的链接代替，该链接通到一个访问 TEXT 的内部节点处)。为了分析的目的，我们可以假定如所示那样存在外节点。

由于通过 Patricia 进行的成功查找结束于外节点处，故在一次随机的成功查找中所作的二进探查的平均数将是外部路径长度除以 N。如果我们对于外节点，像上边那样构造生成函数 $b(z)$，则这将是 $b'(1)/b(1)$。通过 Patricia 的一次不成功的查找，也结束在一个外部节点处，但级 l 上的外节点均以概率 2^{-l} 加权，所以二进位探查的平均数是 $\frac{1}{2}b'\left(\frac{1}{2}\right)$。例如，在图 36 中，我们有 $b(z)=3z^3+8z^4+3z^5+2z^6$；平均说来，每次成功的查找有 $4\frac{1}{4}$ 个二进位探查，每次不成功的查找有 $3\frac{25}{32}$ 个。

设 $h_n(z)$ 是由 n 个外节点，利用一致分布的键码构造成的一株 Patricia 树的“平

均"$b(z)$。递推关系

$$h_n(z)=2^{1-n}\sum_k\binom{n}{k}h_k(z)(z+\delta_{kn}(1-z)),\ h_0(z)=0,\ h_1(z)=1 \quad (15)$$

似乎没有简单的解。幸而平均的外部路径长度 $h'_n(1)$有一个简单的递推式,因为

$$h'_n(1)=2^{1-n}\sum_k\binom{n}{k}h'_k(1)+2^{1-n}\sum_k\binom{n}{k}k(1-\delta_{kn})=$$

$$n-2^{n-1}n+2^{1-n}\sum_k\binom{n}{k}h'_k(1) \quad (16)$$

由于这有(6)的形式,我们就可以使用已经建立的用来解 $h'_n(1)$的诸方法,其结果恰巧比在一个随机的二进检索结构中对应的二进位的探查数小 n。于是对于随机数据,SKIP 字段为每次成功的查找节省一个二进位探查(见习题 31)。典型的实际数据的冗余性将导致更大的节省。

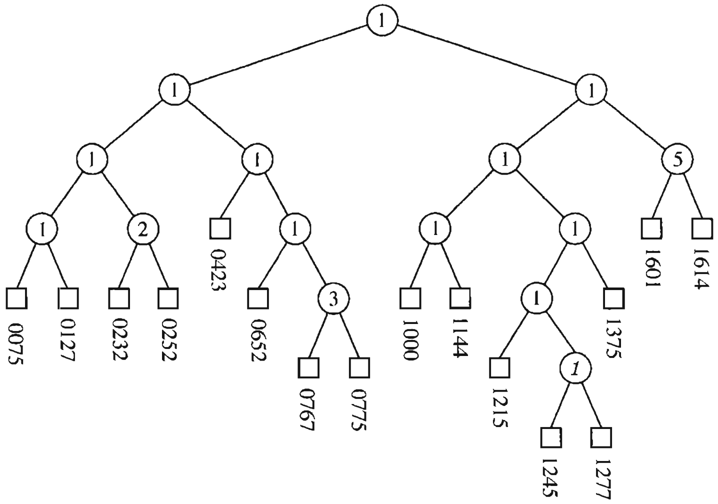


图 36　Patricia 构造代替图 34 的树

当我们试图推算用 Patricia 进行的一次随机的不成功查找的二进位探查平均数时,我们得到递推式

$$a_n=1+\frac{1}{2^n-2}\sum_{k<n}\binom{n}{k}a_k,\ 对于\ n\geqslant 2;\ a_0=a_1=0 \quad (17)$$

这里,$a_n=\frac{1}{2}h'_n\left(\frac{1}{2}\right)$。这不是我们已经研究过的任何递推式的形式,而且也不容易把它转换成过去见过的那种递推式。在 5.2.2 小节介绍的 Mellin 转换理论以及在

那里引用的参考文献,提供了处理有一个数字特征的递推式的高级方法。事实上,已经证明了这个解包含贝努利数:

$$\frac{na_{n-1}}{2} - n + 2 = \sum_{k=2}^{n-1} \binom{n}{k} \frac{B_k}{2^{k-1} - 1} \qquad 对于\ n \geqslant 2 \tag{18}$$

这个公式大概是我们需要解决的最难的渐近问题;习题 34 中的解,是对我们以前做过的许多事情的有益的回顾,并带有某些稍微不同的意味。

分析的综述 作为本节所有复杂的数学推导的结果,下列事实也许是最值得注意的。

a)为把 N 个随机键码存进一个 M 进检索结构中所需要的节点数(检索结构中的分支终止于键数$\leqslant s$ 的子文件)近似地是 $N/(s\ln M)$。这个近似对于很大的 N,小的 s,以及小的 M 是正确的。由于一个检索结构包含 M 个链接字段,故我们若选择 $s=M$,则仅需要大约 $N/\ln M$ 个链接字段。

b)在所有考虑过的方法中,一次随机查找期间考察的数字或字符个数,近似为 $\log_M N$。当 $M=2$ 时,各种分析给了我们下列更精确的二进位探查数的近似值:

	成功的	不成功的
检索结构查找	$\lg N + 1.33275$	$\lg N - 0.10995$
数字树查找	$\lg N - 1.71665$	$\lg N - 0.27395$
Patricia	$\lg N + 0.33275$	$\lg N - 0.31875$

(这些近似都可用基本数学常数来表达,例如 0.31875 代表$(\ln\pi-\gamma)/\ln 2-1/2$。)

c)这里的"随机"数据意味着 M 进数字是一致分布的,就好像诸键是 0~1 之间以 M 进记法来表达的实数那样。数字查找方法同键码进入文件的次序无关(算法 D 除外,它是惟一对次序稍微敏感的);但它们对于数字的分布非常敏感。例如,如果 0 的二进位比 1 的二进位更普遍,则这株树将变得比上面分析的随机数据的情况显得更为倾斜(习题 5.2.2-53 探讨了一个例子,说明当数据如此偏向时发生的情况)。

习 题

1.[*00*] 如果一株树有叶,则一个检索结构有什么?

2.[*20*] 利用算法 T 的约定,试设计把一个新的键码插入到一个 M 进检索结构中去的算法。

3.[*21*] 利用算法 T 的约定,试设计从一个 M 进检索结构删去一个键码的算法。

▶4.[*21*] 表 1 中的 360 个项的大多数都是空白(空链接)。但是我们可以通过使一些空的项同非空的项进行重叠,把这张表压缩成 49 个项如下:

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Entry		(10)		WAS	THAT	(11)	OF	BE	THE	HIS	WHICH	WITH	THIS		(12)	ON	I	HE	A	OR	(2)	(3)	TO	HAD	

Position	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49
Entry	(4)	BUT	(5)	(6)	FOR	BY	IN	FROM	AND	NOT	(7)	HER	ARE	IS	IT	AS	AT	(8)		HAVE	(9)		YOU	

(表 1 的节点(1),(2),…,(12)分别在这个压缩表内的位置 20,19,3,14,1,17,1,7,3,20,18,4 处开始。)

证明,如果以压缩的表代替表 1,则程序 T 仍将有效,但不十分快。

▶5.[*M26*] (Y.N.Patt)　图 31 的诸树,在每族中它们的的字母都是按字符顺序排列的。这种顺序不是必要的,如果我们在构造像(2)那样的二叉树表示之前,重新安排族内节点的顺序,则可得到更快的查找。从这个观点看,图 31 的什么样的重新排列是最优的?(利用图 32 的频率假定,并且去找这样的森林,即当它表示成一株二叉树时,它使成功的查找时间极小化。)

6.[*15*] 如果通过算法 D 以递增次序插入 15 个 4 位二进键码 0001,0010,0011,…,1111,则将得到什么样的数字查找树(在根处以 0001 开始,而后进行 14 次插入)?

▶7.[*M26*] 如果以另一种次序插入习题 6 的 15 个键码,则我们得到一株不同的树。在这些键码的所有 15! 个可能的排列当中,哪一个是*最坏的*? 所谓最坏指的是它产生具有最大内部路径长度的一株树。

8.[*20*] 考虑下列对于算法 D 的修改,它有消去变量 K' 的作用:在步骤 D2 中在两个位置上都把"K'"改成"K",并从步骤 D1 删去操作"$K'\leftarrow K$"。试问得到的算法对于查找和插入是否仍将正确?

9.[*21*] 写出算法 D 的一个 MIX 程序,并把它同程序 6.2.2T 作比较。你可以使用诸如 SLB(左移的二进位 AX)。JAE(如果 A 为偶则转移)等二进操作;你也可以使用习题 8 的思想,如果它有帮助的话。

10.[*23*] 给定一个文件,其中所有键码都是 n 位二进数,并给定一个查找变元 $K=b_1b_2\cdots b_n$,假设我们要来求 k 的这样一个极大值,使得在这个文件中存在一个键码,它以二进位模式 $b_1b_2\cdots b_k$ 开始。

如果把这个文件表示成

a)一株二进查找树(算法 6.2.2T);

b)一个二进检索结构(算法 T);

c)一个二进数字查找树(算法 D),则我们如何能有效地进行?

11.[*21*] 如果不作改变,试问算法 6.2.2D 是否可用来从一株数字查找树删去一个节点?

12.[*25*] 在由算法 D 构造的一株随机数字查找树中删去一个随机元素之后,得到的树是否仍是随机的(参考习题 11 和定理 6.2.2H)?

13.[*20*] (*M 进数字查找*) 说明算法 T 和 D 怎样才能被组合成为一个扩充的算法,即当 $M=2$ 时它实际上同算法 D 一样。如果对于 $M=30$ 使用你的算法,则对表 1 将作什么改动?

▶14.[*25*] 试设计一个有效的算法,在算法 P 成功地结束之后,它能接着做下去,以确定出 K 出现于 TEXT 中的所有位置。

15.[28] 试设计一个有效的算法,它能用来构造为 Patricia 所使用的树,或者用来把新的 TEXT 访问插入到一株现存的树当中,你的插入算法至多只能访问 TEXT 阵列两次。

16.[22] 为什么对 Patricia 作出这样的限制,即任何键码不能是另一个键码的前缀,是合乎要求的?

17.[M25] 通过推广习题 5.2.2-36 的技术,借助二项式变换,找出一种表达递推式

$$x_0 = x_1 = 0; \quad x_n = a_n + m^{1-n}\sum_k \binom{m}{k}(m-1)^{n-k}x_k, \quad n \geqslant 2$$

解的方式。

18.[M21] 借助类似于习题 5.2.2-38 中定义的函数 U_n 和 V_n,利用习题 17 的结果,表达(4)和(5)的解。

19.[HM23] 对于固定的 $s \geqslant 0$ 和 $m > 1$,求当 $n \to \infty$ 时函数

$$K(n,s,m) = \sum_{k \geqslant 2}\binom{n}{k}\binom{k}{s}\frac{(-1)^k}{m^{k-1}-1}$$

的渐近值到 $O(1)$。[在习题 5.2.2-50 中已解决了 $s=0$ 的情况,而在习题 5.2.2-48 中已经解决了 $s=1, m=2$ 的情况]。

▶20.[M30] 如果在 M 进检索结构存储中,当我们达到键数 $\leqslant s$ 的子文件时,即使用顺序查找(算法 T 是 $s=1$ 时的特殊情况)。试应用上面一些习题的结果来分析

a)检索结构节点的平均数;

b)在一次成功的查找中数字或字符探查的平均数;以及

c)在一次成功的查找中所作的平均比较次数。

对于固定的 M 和 s,把你的答案叙述成当 $N \to \infty$ 时的渐近公式;对于(a)的答案应该精确到 $O(1)$ 的范围内,而对(b)和(c)的答案应该精确到 $O(N^{-1})$ 的范围内。[当 $M=2$ 时,这个分析也可应用于修正的基数交换排序上,在这个排序中,其大小 $\leqslant s$ 的子文件是通过插入排序的。]

21.[M25] 在含有 N 个键码的一个随机 M 进检索结构中,有多少个节点在表的项 0 中有一个空的指针?(例如,在表 1 的 12 个节点中,有 9 个在“␣”位置中有空指针。本题中的“随机”像通常一样意味着键码的字符一致地分布在 0 与 $M-1$ 之间。)

22.[M25] 在含有 N 个键码的一个 M 进检索结构中对于 $l=0,1,2,\cdots$,有多少检索结构节点位于级 l 上。

23.[M26] 在含有 N 个随机键码的一个 M 进检索结构中,在一次不成功的查找期间,平均作多少个数字探查?

24.[M30] 考虑一个 M 进检索结构,它已经表示作一片森林(参见图 31)。试找出对于下列两项精确的和渐近的表达式:

a)在森林中节点的平均数;

b)在一次随机的成功查找中执行的“P←RLINK(P)”的平均次数。

▶25.[M24] 在本节中对于渐近值的数学推导已经十分困难,同时涉及复变理论,这是因为希望得到比渐近特性的第一项还要多的结果(而第二项实质上是复杂的)。本题的目的是证明,对于导出某些较弱的结果,初等方法已够用了。

a)通过归纳法证明,(4)的解满足 $A_N \leqslant M(N-1)/(M-1)$。

b)设 $D_N = C_N - NH_{N-1}/\ln M$,其中 C_N 由(5)定义。证明 $D_N = O(N)$,因此 $C_N = N\log_M N + O(N)$。[提示:用(a)和定理 1.2.7A]。

26.[23] 通过手算,确定无穷乘积

$$\left(1-\frac{1}{2}\right)\left(1-\frac{1}{4}\right)\left(1-\frac{1}{8}\right)\left(1-\frac{1}{16}\right)\cdots$$

的值，精确到五位十进数。[提示：参见习题 5.1.1-16。]

27.[*HM31*] 如同由(14)给出的那样，$\overline{C}_N$ 精确到 $O(1)$ 的渐近值是什么？

28.[*HM26*] 对一般的 $M\geqslant 2$，当在一株随机的 M 进数字查找树中进行查找时，试求数字探查的渐近平均次数，既考虑成功的也考虑不成功的查找，并给出你的答案，精确到 $O(N^{-1})$。

29.[*HM40*] 在一株 M 进数字查找树中，所有的 M 链接均为空的那些节点的渐近平均数是多少？(我们可以通过消去这样的节点来节省内存空间；参见习题 13。]

30.[*M24*] 证明在(15)中定义的 Patricia 的生成函数可以表达成相当吓人的形式

$$n\sum_{m\geqslant 1} z^m\left(\sum_{\substack{a_1+\cdots+a_m=n-1\\ a_1,\cdots,a_m\geqslant 1}}\binom{n-1}{a_1,\cdots,a_m}\frac{1}{(2^{a_1}-1)(2^{a_1+a_2}-1)\cdots(2^{a_1+\cdots+a_m}-1)}\right)$$

[于是，如果对 $h_n(z)$ 有一个简单的公式，则我们将有能力来简化这个相当笨拙的表达式。]

31.[*M21*] 解递推式(16)。

32.[*M21*] 在具有 $N-1$ 个内节点的一株随机 Patricia 树中，所有的 SKIP 字段之和的平均值是什么？

33.[*M30*] 证明(18)是对递推式(17)的一个解。[提示：考虑生成函数 $A(z)=\sum_{n\geqslant 0}a_n z^n/n!$。]

34.[*HM40*] 本题的目的是求(18)的渐近特性。

a)证明，如果 $n\geqslant 2$

$$\frac{1}{n}\sum_{2\leqslant k<n}\binom{n}{k}\frac{B_k}{2^{k-1}-1}=\sum_{j\geqslant 1}\left(\frac{1^{n-1}+2^{n-1}+\cdots+(2^j-1)^{n-1}}{2^{j(n-1)}}-\frac{2^j}{n}+\frac{1}{2}\right)$$

b)证明(a)中的被加数近似于 $1/(e^x-1)-1/x+1/2$，其中 $x=n/2^j$；得到的和等于原来的和 $+O(n^{-1})$。

c)证明

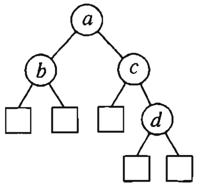
$$\frac{1}{e^x-1}-\frac{1}{x}+\frac{1}{2}=\frac{1}{2\pi i}\int_{-\frac{1}{2}-i\infty}^{-\frac{1}{2}+i\infty}\zeta(z)\Gamma(z)x^{-z}dz,\ 对于实\ x>0$$

d)因此这个和等于

$$\frac{1}{2\pi i}\int_{-\frac{1}{2}-i\infty}^{-\frac{1}{2}+i\infty}\frac{\zeta(z)\Gamma(z)n^{-z}}{2^{-z}-1}dz+O(n^{-1})$$

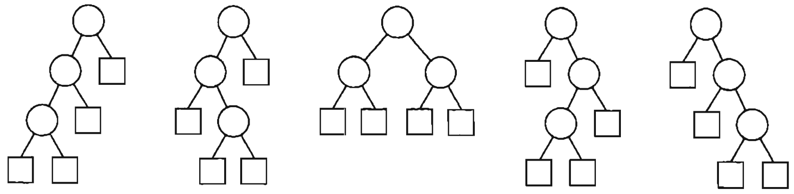
计算这个积分。

▶35.[*M20*] 在五个键上的 Patricia 树将是



且有如图所示的 SKIP 字段 a,b,c,d 的概率是多少？(假定诸键码都有独立的随机二进数，把答案作为 a,b,c,d 的函数给出。)

36.[*M25*] 兹有具有三个内节点的五株二叉树。如果我们考虑在各种算法下,对于随机数据这些树中的每一株作为查找树出现的频率,则我们发现下列不同的概率:



查找树 (算法 6.2.2T)	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{6}$	$\frac{1}{6}$
数字树查找 (算法 D)	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{2}$	$\frac{1}{8}$	$\frac{1}{8}$
Patricia (算法 P)	$\frac{1}{7}$	$\frac{1}{7}$	$\frac{3}{7}$	$\frac{1}{7}$	$\frac{1}{7}$

(注意数字查找树往往比其它树更经常地成为平衡的。)在习题 6.2.2-5 中,我们发现在树查找算法中,一株树的概率是$\prod(1/s(x))$,其中乘积对所有内部节点 x 进行,而且 $s(x)$是在以 x 为根的子树中内节点的个数。在(a)算法 D;(b)算法 P 的情况下,试求一株树的概率的类似公式。

▶37.[*M22*] 考虑在级 l 上具有 b_l 个外节点的一株二叉树。正文发现在数字查找树中,不成功的查找的运行时间不直接同外部路径长度$\sum lb_l$ 有关,而是实质上同修正过的外部路径长度$\sum lb_l2^{-l}$成正比。证明或否定:对于具有 N 个外节点的所有树来说,最小的修正过的外部路径长度出现的条件是,所有外节点都出现在至多两个相邻的级上[参见习题 5.3.1-20]。

38.[*M40*] 给定 α 和β,在习题 37 的意义下,试建立一种算法,来找使 $\alpha\cdot$(内部路径长度)+$\beta\cdot$(修正过的外部路径长度)取极小值的 n 个节点的树。

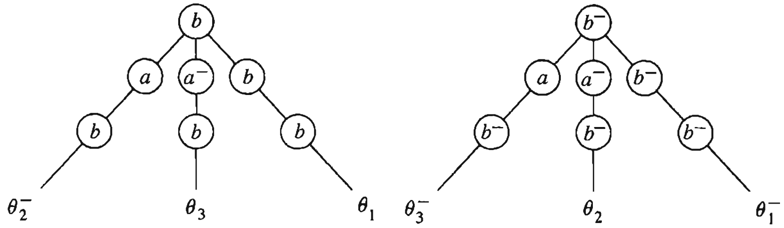
39.[*M43*] 建立一个算法,来找类似于 6.2.2 小节所考虑的最优二分查找树的最优数字查找树。

▶40.[*25*] 设 $a_0a_1a_2\cdots$是一个周期二进序列,且对于所有 $k\geqslant0$,皆有 $a_{N+k}=a_k$。证明有一种方法以 $O(N)$个内存单元来表示这种类型的任何固定序列,使得下列操作可以在仅仅 $O(n)$步内完成:给定任何二进型式 $b_0b_1\cdots b_{n-1}$,确定这个形式在周期中出现的频繁程度(即,问有多少 p 值存在且同时满足下列两个条件:$0\leqslant p<N$,且对于 $0\leqslant k<n$,有 $b_k=a_{p+k}$)。这个型式的长度 n 以及型式本身也是变量。假定每一个内存单元都足够大来存放 $0\sim N$ 之间的任何整数。[提示:参见习题 14]

41.[*HM28*] 这是对群论的一个应用。设 G 是字母$\{a_1,\cdots,a_n\}$上的自由群,即,所有串 $\alpha=b_1\cdots b_r$ 的集合,其中每个 b_i 是 a_j 或 a_j^{-1} 之一,且没有相邻的对 $a_ja_j^-$ 或 $a_j^-a_j$ 出现。α 的逆是$b_r^-\cdots b_1^-$,两个这样的串的相乘方法是连接它们并消去相邻的互逆的对。设 H 是通过串$\{\beta_1,\cdots,\beta_p\}$生成的 G 的子群,即,可以写成诸 β 及它们的逆的乘积的 G 的所有元素的集合。按照(Jakob Nielsen)一个著名定理(参见 Marshall Hall, The *Theory of Groups* (纽约:Macmillan,1959),第 7 章)。我们总可以求 H 的生成元$\theta_1,\cdots,\theta_m$,且 $m\leqslant p$,它们具有如下性质:诸 θ_i 的中间字(或者如果它的长度为偶数,则至少为 θ_i 的两个中间字符之一)在表达式 $\theta_i\theta_j^e$ 或$\theta_j^e\theta_i$ 中绝不被抵销,这里 $e=\pm1$,除非 $j=i$ 且$e=-1$。这个性质意味着,有一个简单的算法用于检测 G 的一个任意的元素是否在H 中:利用 $2n$ 个字母$a_1,\cdots,a_n,a_1^-,\cdots,a_n^-$ 在一个面向字符的查找树中记录 $2m$ 个键θ_1,

$\cdots,\theta_m,\theta_1^-,\cdots,\theta_m^-$。设 $\alpha=b_1\cdots b_r$ 是 G 的一个给定的元素;如果 $r=0$,则 α 显然在 H 中。否则查找 α,求能匹配一个键码的最长前缀 $b_1\cdots b_k$。如果有一个以上以 $b_1\cdots b_k$ 开始的键码,则 α 不在 H 中;否则设惟一的这样的键码是 $b_1\cdots b_kc_1\cdots c_l=\theta_i^e$,以 $\theta_i^{-e}\alpha=c_l^-\cdots c_1^-b_{k+1}\cdots b_r$ 代替 α。如果这个 α 的新值比旧的更长(即如果 $l>k$),则 α 不在 H 中;否则对 α 的新值重复此过程。Nielsen 性质意味着这个算法总能结束。如果 α 最终归结成空串,则我们可以把原来的 α 的表示重新构造成为诸 θ 的一个乘积。

例如,设 $\{\theta_1,\theta_2,\theta_3\}=\{bbb,b^-a^-b^-,ba^-b\}$ 且 $\alpha=bbabaab$,森林



可用于上述算法来导出 $\alpha=\theta_1\theta_3^-\theta_1\theta_3^-\theta_2^-$。给定诸 θ 作为你的程序的输入试实现这个算法。

42.[*23*] (前压缩和后压缩)　当二进键码的一个集合被用作下标以分划一个较大的文件时,我们不必存储完全的键码。例如,如果使用图 34 的 16 个键码,则只要已经给出足够位的数字来惟一地识别它们,就可以在右边截断这些键码:0000,0001,00100,00101,010,…,1110001。这些截断了的键码可用来把一个文件划分成 17 部分,其中,例如第五部分由所有的以 0011 或 010 开始的键码组成,而最后部分包含以 111001,11101 或 1111 开始的所有键码。在截断后的键码中,如果我们去掉所有同上一个键码相同的前导数字,则可以表示得更紧凑:0000,◇◇◇1,◇◇100,◇◇◇◇1,◇10,…,◇◇◇◇◇◇1。跟随◇后边的二进位总是 1,所以它可以去掉。一个大的文件将有许多◇,而且我们仅仅需要存储◇的个数和随后的二进位的值。

试证明,在压缩的文件中,除开◇和跟随◇的二进位 1 外,二进位的总数总等于键码的二进检索结构中的节点数。

(因此,在整个下标中,这样的二进位的平均总数大约是 $N/\ln 2$,即每个键码仅 1.44 个二进位。这项压缩技术是由 A.Heller 和 R.L.Johnsen 告知作者的。由于我们仅需表示检索结构,因此还可能作进一步压缩;参见定理 2.3.1*A*。)

43.[*HM42*] 试分析有 N 个键码和如同在习题 20 中一样有切断参数 s 的一个随机 M 进检索结构的高度。(当 $s=1$ 时,这是在 M 进字母表中 N 个长的随机字的最长公共前缀的长度。)

▶44.[*30*] (J.L.Bentley 和 R.Sedgewick)　试剖析检索结构的一个三进表示,其中左右链接对应于(2)的水平分支而中间链接对应于向下的分支。

45.[*M25*] 如果通过习题 15 的算法把图 33 的七个关键字以随机次序插入,问得到所示的树的概率是多少?

6.4　散　列

至今我们已经考虑的查找方法,是以给定的变元 K 同表中的键码作比较,或使用它的数字,来支配一个转移过程为基础的。第三种可能性是避免来回搜查,这就是通过对 K 作某种算术运算,计算一个函数 $f(K)$,它是 K 的位置和表中的相关数

据。

例如，再次考虑 31 个英文字的集合，在 6.2.2 小节和 6.3 节中我们已经对于这些英文字施以各种查找策略。表 1 示出一个短的 MIX 程序，它把这 31 个键码的每一个都变成为在 −10 和 30 之间惟一的数 $f(K)$。如果我们把这个 MIX 程序所用的方法同已经考虑的其它方法(例如，二分查找，最优树查找，检索结构存储，数字树查找)作比较，则发现，从空间和速度两方面看，它都是更好的，只有二分查找使用的内存稍微少些。事实上，对于图 12 的频率数据，使用表 1 的程序，一次成功的查找的平均时间，大约仅 $17.8u$，而且为存储 31 个键码仅需要 41 个表单元。

可惜，发现这样的函数 $f(K)$并不是很容易的。从一个 31 个元素的集合到一个 41 元素的集合有 $41^{31}\approx 10^{50}$种可能的函数，而且它们当中仅仅 $41\cdot 40\cdot \cdots \cdot 11 = 41!/10!\approx 10^{43}$个能对每个变元给出不同的值；于是每一千万个函数当中大约仅仅有一个是适用的。

避免重复值的函数是异常稀少的，甚至对于一个相当大的表也是这样。例如，著名的"生日诗论"断言，如果有 23 个以上的人在一个房间里，则他们当中两个人有相同的出生月日的可能性很大。换言之，如果我们选择一个随机函数，它映射 23 个键码到大小为 365 的一张表中，则两个键码不映射到同一个单元的概率仅仅是 0.4927(少于二分之一)。这一结果的怀疑者们不妨在他们参加的下一次大型集会上查查各人的生日有无相同的。

[生日悖论在 1930 年代曾由数学家们非正式地讨论过，但它的起源不清楚。参见(I.J.Good,*Probability and the Weighing of Evidence* (Griffin,1950), 38,也请参见(R.von Mises), *Istanbul Üniversitesi Fen Fakültesi Mecmuasl* **4**(1939),145～163,以及(W.Feller),*An Introduction to Probability Theory* (纽约:Wiley,1950),2.3 节]

另一方面，表 1 中所用的方法是相当灵活的[参考 M.Greniewski 和 W.Turski, *CACM* **6** (1963),322～323]，而且对于一张中等规模的表，在进行了大约一天的工作之后，就能找到一个适当的函数。事实上，解决像这样的难题是颇为有趣的。许多人都曾参加过有关适用技术的讨论，例如，包括 R.Sprugnoli,*CACM* **20** (1977), 841～850,**22**(1979),104,553;R.J.Cichelli, *CACM* **23**(1980),17～19;T.J.Sager, *CACM* **28** (1985), 523 ～ 532, **29** (1986), 557; B.S.Majewski, N.C.Wormald, G.Havas,以及 Z.J.Czech,*Comp.J.* **39** (1996),547～554;Czech,Havas 和 Majewski, *Theoretical Comp. Sci.* **182**(1997),1～143。关于完全散列函数的理论极限，也请参见 J.Körner 和 K.Marton, *Europ.J. Combinatorics* **9** (1988),523～530。

当然，这个方法有一个严重的缺陷，因为这张表的内容必须是预先知道的；增多一个键码都可能使一切崩溃，以致几乎需要从 0 开始。如果我们放弃惟一性的思想，允许不同的键码产生相同的 $f(K)$值，而且在计算了 $f(K)$之后，再使用一种专门的方法解决任何二义性，就可以得到一个更多样化的方法。

这些考虑导致了通常称做散列或散列存储技术的一类流行的查找方法。动词"散列"意味着把某些事物剁碎或把它弄得乱七八糟；散列的思想是把键码的某些内

表 1　把一组键转换成惟一的地址

	指令		A	AND	ARE	AS	AT	BE	BUT	BY	FOR	FROM	HAD	HAVE	HE	HER	HIS	I	IN	IS	IT	NOT	OF	ON	OR	THAT	THE	THIS	TO	WAS	WHICH	WITH	YOU
			给定一个特殊的键 K，在执行了这条指令后 rI1 的内容																														
	LD1N	K(1:1)	−1	−1	−1	−1	−1	−2	−2	−2	−6	−6	−8	−8	−8	−8	−8	−9	−9	−9	−9	−15	−16	−16	−16	−23	−23	−23	−23	−26	−26	−26	−28
	LD2	K(2:2)	−1	−1	−1	−1	−1	−2	−2	−2	−6	−6	−8	−8	−8	−8	−8	−9	−9	−9	−9	−15	−16	−16	−16	−23	−23	−23	−23	−26	−26	−26	−28
	INC1	−8,2	−9	6	10	13	14	−5	14	18	2	5	−15	−15	−11	−11	−7	−17	−2	5	6	−7	−18	−9	−5	−23	−23	−23	−15	−33	−26	−25	−20
	J1P	* +2	−9	6	10	13	14	−5	14	18	2	5	−15	−15	−11	−11	−7	−17	−2	5	6	−7	−18	−9	−5	−23	−23	−23	−15	−33	−26	−25	−20
	INC1	16,2	7	·	·	·	·	16	·	·	·	·	2	2	10	10	18	−1	29	·	·	25	4	22	30	1	1	1	17	−16	−2	0	12
	LD2	K(3:3)	7	6	10	13	14	16	14	18	2	5	2	2	10	10	18	−1	29	5	6	25	4	22	30	1	1	1	17	−16	−2	0	12
	J2Z	9F	7	6	10	13	14	16	14	18	2	5	2	2	10	10	18	−1	29	5	6	25	4	22	30	1	1	1	17	−16	−2	0	12
	INC1	−28,2	·	−18	−13	·	·	·	9	·	−7	−7	−22	−1	·	1	12	·	·	·	·	20	·	·	·	−26	−22	−18	·	−22	−21	−5	8
	J1P	9F	·	−18	−13	·	·	·	9	·	−7	−7	−22	−1	·	1	12	·	·	·	·	20	·	·	·	−26	−22	−18	·	−22	−21	−5	8
	INC1	11,2	·	−3	3	·	·	·	·	·	23	20	−7	35	·	·	·	·	·	·	·	·	·	·	·	−14	−6	2	·	11	−1	29	·
	LDA	K(4:4)	·	−3	3	·	·	·	·	·	23	20	−7	35	·	·	·	·	·	·	·	·	·	·	·	−14	−6	2	·	11	−1	29	·
	JAZ	9F	·	−3	3	·	·	·	·	·	23	20	−7	35	·	·	·	·	·	·	·	·	·	·	·	−14	−6	2	·	11	−1	29	·
	DEC1	−5,2	·	·	·	·	·	·	·	·	·	9	·	15	·	·	·	·	·	·	·	·	·	·	·	−10	·	−2	·	·	−5	11	·
	J1N	9F	·	·	·	·	·	·	·	·	·	9	·	15	·	·	·	·	·	·	·	·	·	·	·	−10	·	−2	·	·	−5	11	·
	INC1	10	·	·	·	·	·	·	·	·	·	19	·	25	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	21	·
9H	LDA	K	7	−3	3	13	14	16	9	18	23	19	−7	25	10	1	12	−1	29	5	6	20	4	22	30	−10	−6	−2	17	11	−5	21	8
	CMPA	TABLE,1	7	−3	3	13	14	16	9	18	23	19	−7	25	10	1	12	−1	29	5	6	20	4	22	30	−10	−6	−2	17	11	−5	21	8
	JNE	FAILURE	7	−3	3	13	14	16	9	18	23	19	−7	25	10	1	12	−1	29	5	6	20	4	22	30	−10	−6	−2	17	11	−5	21	8

容打乱,并且使用这种部分的信息作为查找的基础。我们计算一个散列函数 $h(K)$ 而且使用这个值作为查找的开始地址。

生日诗论告诉我们,很可能有不同的键码 $K_i \neq K_j$,散列成相同的值 $h(K_i) = h(K_j)$。这种情况称作冲突,而且若干有趣的方法已被想出来处理冲突问题。为了使用一个散列表,一个程序员必须作两个几乎独立的决定:必须选择一个散列函数 $h(K)$ 和解决冲突的方法。现在我们依次考虑这个问题的两个方面。

散列函数 为使事情更加明显,我们在这一节始终假定,散列函数最多取 M 个不同的值,且对于所有的键码 K

$$0 \leqslant h(K) < M \tag{1}$$

实际上出现在真正文件中的键码,通常都有大量的冗余性;我们必须小心寻找一个散列函数,它拆散几乎相同的成群的键码,以便减少冲突的个数。

从理论上说,不可能定义这样一个散列函数,它由实际文件中的非随机数据建立随机的数据。但实际上,通过使用如同在第三章讨论的简单算术运算,不难产生出随机数据的一个相当好的拟态。而且事实上,通过进一步利用实际数据的非随机性质,来构造出一个比真正随机的键码还少产生冲突的散列函数,我们常常甚至可以做得更好。

例如,考虑在一台十进制计算机上的 10 位数字键码的情况。一个可考虑的散列函数是,比如说,命 $M = 1000$,以及命 $h(K)$ 是从 20 位数字的乘积 $K \times K$ 的接近于中间的某处选出的三位数字。这似乎将产生出在 000～999 之间相当好的散开的值,而且冲突概率较低。事实上,通过实际数据的实验表明,假定键码中没有大量的前导或后尾的 0,这个“平方取中”的方法并不坏;但恰如我们在第三章中发现的,平方取中的方法不是一个特别好的随机数生成程序。还有更安全和更明智的处理方法。

对典型文件所作的广泛实验已经表明,有两大类散列函数十分好。一是以除法为基础的,而另一是以乘法为基础的。

除法方法特别容易;我们只需使用模 M 的余数:

$$h(K) = K \bmod M \tag{2}$$

在这种情况下,M 的某些值显然比其它值要好得多。例如,如果 M 是一个偶数,则当 K 是偶数时 $h(K)$ 将是偶数,当 K 是奇数时,$h(K)$ 也是奇数,在许多文件中这将导致相当大的偏向。若 M 是计算机基数的乘方,那将是更坏的,因为 $K \bmod M$ 将仅是 K 的最低位上的一些数字(而同其它数字无关)。类似地,我们可以论证,M 大概也不应是 3 的倍数;因为如果诸键码都是字符的,则两个彼此仅仅是字母排列不同的键码,在数值上可能仅仅差 3 的一个倍数(这是由于 $2^{2n} \bmod 3 = 1$ 和 $10^n \bmod 3 = 1$ 而出现的现象)。一般地说,我们要避免能整除 $r^k \pm a$ 的 M 值,其中 k 和 a 是较小的数且 r 是字符集合的基数(通常 $r = 64, 256$ 或 100),因为对这样一个 M 值求模的余数,在很大程度上只是键码中数字的简单叠加。这样一个考虑提示我们选择 M 为一个素数使得对于小的 k 和 a,$r^k \not\equiv \pm a \pmod{M}$。已经发现,这种选择在大

多数情况下都是十分令人满意的。

例如，在 MIX 计算机上，我们可以选择 $M=1009$，通过序列

```
LDX    K           rX ← K
ENTA   0           rA ← 0                      (3)
DIV    =1009=      rX ← K mod 1009
```

计算 $h(K)$。

乘法的散列方案也同样容易做出，但要稍难描述些，因为我们必须想像是对分数进行运算而不是对整数。设 w 是计算机字的大小，对于 MIX，它通常是 10^{10} 或 2^{30}；我们可以把整数 A 当作是分数 A/w，这里我们想像小数点位于这个字的左边。这方法就是来选择与 w 互素的某个整数常数 A，而且命

$$h(K)=\left\lfloor M\left(\left(\frac{A}{w}K\right)\bmod 1\right)\right\rfloor \tag{4}$$

在这种情况下，我们通常设 M 在一台二进计算机上是 2 的乘方，于是 $h(K)$ 由乘积 AK 的较低一半的一些前导二进位组成。

在 MIX 代码中，如果我们设 $M=2^m$ 且采用二进制，则乘法散列函数就是

```
LDA    K    rA ← K
MUL    A    rAX ← AK
ENTA   0    rAX ← AK mod w                         (5)
SLB    m    rAX 的左移 m 个二进位
```

现在 $h(K)$ 出现在寄存器 A 中。由于 MIX 的乘法和移位指令相当慢，故这个序列恰巧花费和(3)一样长的计算；但是在许多机器上乘法比除法要快得多。

在某种意义下这个方法可以认为是(3)的推广，因为例如我们可以取 A 为 $w/1009$ 的一个近似值；乘以一个常数的倒数，通常要比除以该常数更快些。注意，(5)几乎就是“平方取中”方法，但有一点重要的区别：我们将看到，乘以一个适当的常数有许多可论证的好性质。

乘法方案好的特征之一在于，当在(5)消除寄存器 A 时，不损失信息；在(5)完成之后，仅仅给定 rAX 的内容，即可再次确定 K。原因在于 A 与 w 互素，所以欧几里得算法可用来求一常数 A'，使得 $AA' \bmod w=1$；这意味着 $K=(A'(AK \bmod w))\bmod w$。换言之，如果 $f(K)$ 表示在(5)中的 SLB 指令之前寄存器 X 的内容，则

$$K_1\neq K_2 \quad 意味着 \quad f(K_1)\neq f(K_2) \tag{6}$$

当然，$f(K)$ 在 0 到 $w-1$ 的范围内取值，所以它不像一个散列函数那样好，但作为一个扰乱函数是非常有用的。所谓扰乱函数就是满足(6)和趋向于把键码随机化的一个函数。如果键码的次序无关重要，则在同 6.2.2 小节的树查找函数相联系时，这样一个函数可能是非常有用的，因为当键码以递增的次序进入树时，它消除了退化的危险性。(参见习题 6.2.2—10)。如果实际的键码的二进位有偏向时，则一个扰乱函数在同 6.3 节的数字树查找算法相联系时也是有用的。

乘法散列方法的另一个特征是，它很好地利用了在许多文件中存在的非随机性。实际的键码集合通常有一种算术级数的倾向，其中 $\{K,K+d,K+2d,\cdots,K+$

td}全都出现在文件中；例如考虑像{PART1, PART2, PART3}或者{TYPEA, TYPEB, TYPEC}这样的字符名称。乘法散列方法把一个算术级数转换成不同散列值的近似的算术级数 $h(K)$, $h(K+d)$, $h(K+2d)$, …，同时减少在随机状态下我们所预期的冲突个数。除法方法也有同样的性质。

图 37 在一种特别有趣的情况下，说明了乘法散列的这个方面。假定 A/w 近似

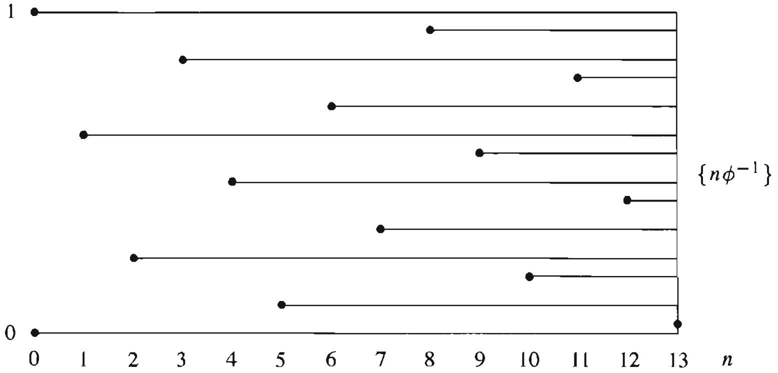


图 37　斐波那契散列

地是黄金比 $\phi^{-1}=(\sqrt{5}-1)/2=0.6180339887$；则相继的值 $h(K)$, $h(K+1)$, $h(K+2)$……和相继的散列值 $h(0)$, $h(1)$, $h(2)$, …有相同的特性，这提示了下面的实验：由线段[0..1]开始，逐次地标出点 $\{\phi^{-1}\}$, $\{2\phi^{-1}\}$, $\{3\phi^{-1}\}$, …，其中 $\{x\}$ 代表 x 的小数部分（即 $x-\lfloor x\rfloor$，或 $x \bmod 1$）。如同在图 37 中所示，这些点彼此之间分得很开；事实上，每个最新增加的点都落入最大的剩下的区间之一，而且按黄金比分割它[这个现象首先是由 J. Oderfeld 猜测，并且由 S. Swierczkowski 证明的，*Fundamenta Math*. **46** (1958), 187～189]。在这个证明中，斐波那契数起着重要作用]。

黄金比的这个值得注意的性质实际上原来是由 Hugo Steinhaus 猜测并且首先由 Vera Turán Sós 证明的[*Acta Math. Acad. Sci. Hung*. **8**(1957), 461～471; *Ann. Univ. Sci. Budapest. Eötvös Sect. Math*. 1(1958), 127～134]：

定理 S *设 θ 是任意无理数。当把点 $\{\theta\}$, $\{2\theta\}$, …, $\{n\theta\}$ 放置在线段[0..1]中时，形成的 $n+1$ 个线段至多有三种不同的长度。而且，下一点 $\{(n+1)\theta\}$ 将落在最大的现存线段之一中。* ▌

于是，点 $\{\theta\}$, $\{2\theta\}$, …, $\{n\theta\}$ 非常均匀地散列在 0～1 之间。如果 θ 是有理的，则同样的定理也成立，但我们必须对长度为 0 的线段给出一个适当的解释，这长度为 0 的线段是在 n 大于或等于 θ 的分母时出现的。定理 S 的一个证明，连同关于这种状态的基础结构的一个详细分析，出现于习题 8 中；可以证明，具有给定长度的诸线段是以先进先出的方式建立和破坏的。当然，某些 θ 比其它的 θ 更好，因为例如接近于 0 或 1 的一个值将以许多小段和一大段开始。习题 9 证明，在 0 与 1 之间的所有数 θ 当中，两个数 ϕ^{-1} 和 $\phi^{-2}=1-\phi^{-1}$ 导致“最一致地分布”的序列。

上面的理论提示了斐波那契散列，其中我们选择常数 A 作为最接近于 $\phi^{-1}w$ 的与 w 互素的整数。例如，如果 MIX 是十进计算机，则取

+	61	80	33	98	87

$A =$ (7)

这个乘数将非常好地撒开像 LIST1，LIST2，LIST3 这样的字符键码。但注意在第四个字符位置中有一个算术级数的情况，如同在键码 SUM1⊔，SUM2⊔，SUM3⊔ 中那样：其效果就像以 $\theta=\{100A/w\}=.80339887$ 代替 $\theta=.6180339887=A/w$ 来使用定理 S 那样。尽管这个 θ 值不完全像 ϕ^{-1} 那样好，但得到的结果仍然是不错的。另一方面，如果这个级数出现在第二个字符位置上，如同在 A1⊔⊔⊔，A2⊔⊔⊔，A3⊔⊔⊔ 中那样，则有效的 θ 是。.9887，而这可能太接近于 1 了。

因此，我们通过用像

$A =$

+	61	61	61	61	61

这样一个乘数代替(7)，可以做得更好；这样一个乘数将分散在任何字符位置中互异的连续的键码序列。不幸的是，这个选择遇到了另外的问题，有点类似于除以 $r^k\pm1$ 时的困难：诸如 XY 和 YX 这样的键码将趋于散列到相同的单元！摆脱这个困难的一种方式，是更仔细地考察定理 S 背后的基础结构；当键码的级数很短时，则起作用的仅仅是 θ 的连分数表示的开头几个部分商，而且小的部分商对应于好的分布性质。因此我们发现 θ 的最好的值位于范围

$$\frac{1}{4}<\theta<\frac{3}{10},\ \frac{1}{3}<\theta<\frac{3}{7},\ \frac{4}{7}<\theta<\frac{2}{3},\ \frac{7}{10}<\theta<\frac{3}{4}$$

之中。可以找到 A 的一个值，使得它的每个字节都位于一个好的范围之中，而且不太接近于其它字节的值或这些值的补，例如，

$A =$ (8)

+	61	25	42	33	71

可以推荐这样一个乘数。（关于乘法散列的这些思想大部分是由 R.W.Floyd 给出的）。

一个好的散列函数应满足下列两个要求。

a)它的计算应该是非常快的。

b)它应该使冲突极小化。

性质(a)和机器有关，而性质(b)则与数据有关。如果诸键码是真正随机的，则我们可以从其中简单地抽出一些二进位，并使用这些二进位作为散列函数；但在实践中，我们几乎总是需要有一个依赖于键码的所有二进位的散列函数，以便满足(b)。

至今我们已经考虑了怎样散列一个字的键码。多字的或可变长的键码可以通过上述方法的多精度扩充来处理，但是一般宜于把几个字组成一个单个的字，然后如上面那样进行单个乘法或除法，这样可以提高速度。这个组合可以通过 mod w 加法或通过在一台二进计算机上的“异或”来完成；这两个操作都有这样一个优点，即它们都是可逆的，也就是，它们都依赖于两个变元的所有二进位，而且由于异或避免了算术溢出，它有时是更可取的。然而，这两种操作是可交换的，因此(X，Y)和

(Y,X)都将散列成相同的地址；G.D.Knott 已经建议在进行加法或异或之前作一个循环移位以避免这个问题。

散列 l 个字符或 l 个字键码 $K=x_1x_2\cdots x_l$ 的一个甚至更好的方法是来计算

$$h(K)=(h_1(x_1)+h_2(x_2)+\cdots+h_l(x_l))\bmod M \tag{9}$$

其中每个 h_j 是一个独立的散列函数。由 J.L.Carter 和 M.N.Wegman 于 1977 年引入的这一思想，当每个 x_j 是单个字符时特别有效，因为然后我们可以使用对于每个 h_j 的一个预先计算的数组。这样的数组使乘法成为毫无必要的，而且如果 M 是 2 的一个乘幂，我们通过以异或代替加法而避免(9)中的除法。因此(9)肯定满足性质(a)。而且，Carter 和 Wegman 证明了如果 h_j 是随机选择的，则不管输入数据如何，性质(b)都将成立(参见习题 72)。

已经提出了许许多多的散列方法，但是还没有证明出这些方法中哪一个比上面叙述的简单的除法和乘法更优越。关于某些其它方法及其用于实际文件时性能的详细统计，见林耀森、袁师德以及(M.Dodd)，*CACM* **14**(1971)，228～239。

在已经试验过的所有其它散列方法中，也许最有趣的是一项以代数编码理论为基础的技术；其思想类似于上面的除法方法，但我们除以一个多项式 modulo 2 而不是除以一个整数(如同在 4.6 节所观察到的，这个操作类似于除法，就如同加法类似于异或那样)。对于这个方法，M 应该是 2 的一个乘方，比如说，$M=2^m$，而且我们利用一个 m 次多项式$P(x)=x^m+p_{m-1}x^{m-1}+\cdots+p_0$。一个 n 个数字的二进键码 $K=(k_{n-1}\cdots k_1k_0)_2$ 可以看作是多项式 $K(x)=k_{n-1}x^{n-1}+\cdots+k_1x+k_0$，而且利用多项式算术运算modulo 2计算余式

$$K(x)\bmod P(x)=h_{m-1}x^{m-1}+\cdots+h_1x+h_0$$

则 $h(K)=(h_{m-1}\cdots h_1h_0)_2$。如果适当地选择 $P(x)$，则这个散列函数即可以保证避免在近乎相等的键码之间的冲突。例如，如果 $n=15$，$m=10$，且

$$P(x)=x^{10}+x^8+x^5+x^4+x^2+x+1 \tag{10}$$

则可以证明，每当 K_1 和 K_2 是两个不同的在少于七个二进位位置中互异的键码时，$h(K_1)$不等于 $h(K_2)$。(关于这个方案的进一步信息见习题 7；当然它更适合于以硬件或微程序设计实现，而不适合于用软件实现。)

当调试一个程序时，使用一个常数散列函数 $h(K)=0$ 通常很方便，因为所有的键码都将被存储在一起；稍后可以换成一个有效的 $h(K)$。

通过“拉链”解决冲突 我们已经注意到，某些散列地址由于为多个键码所共享，可能要出麻烦。解决这个问题的也许最显然的方法，是保持 M 个链接表，对于每个可能的散列码用一个表。每个记录应包括一个 LINK 字段，而且还将有 M 个表头，比如说以从 $1\sim M$ 来进行编号。在散列了键码之后，我们只需在号码为 $h(K)+1$ 的表中进行顺序查找。(参见习题 6.1-2。这一情况非常类似于多重表的插入排序，即程序 5.2.1M)。

图 38 示出了当 $M=9$ 时，对于七个键码的序列(这就是挪威文中的数 1～7)，

$$K = \texttt{EN,TO,TRE,FIRE,FEM,SEKS,SYV} \qquad (11)$$

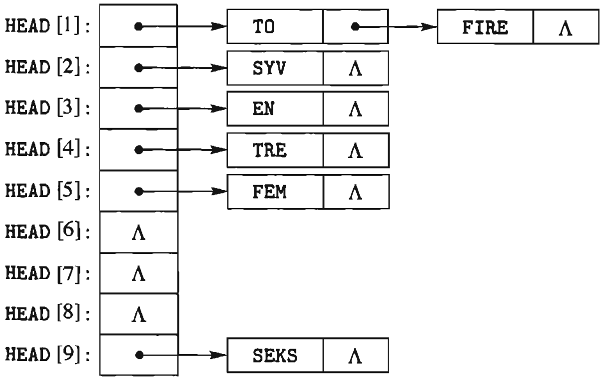


图 38　分开的拉链

这一简单的拉链方案，这些键码的散列码分别为

$$h(K) + 1 = 3,1,4,1,5,9,2 \qquad (12)$$

第一个表有两个元素，而另外有三个表是空的。

拉链十分快，因为这些表都很短。如果把 365 人聚集在一个房间里，则大概将有许多对的人有相同的生日，但是具有任何给定生日的人的平均数将仅仅是 1！一般说来，如果有 N 个键码和 M 个表，则表的平均大小是 N/M；于是散列使顺序查找所需的平均工作数量大约减少了一个因子 M。（习题 34 给出了一个精确的公式。）

这个方法是我们以前讨论的技术的直截了当的组合，所以不需要描述对于拉链散列表的详细算法。以键码的顺序来保持个别的表通常是一个好的想法，它使得插入和不成功的查找都能更快地进行。于是如果我们使表成为递增的，则图 38 的 TO 和 FIRE 节点将互相交换，而且所有的 Λ 链都将被指向一个空记录的指针所代替，该记录的键码为∞（参见算法 6.1T）。或者，我们可以利用 6.1 节中讨论的“自组织文件”的概念；把诸表按各键码最新出现的时间排序，而不是按键码的值排序。

为了保证速度，我们将乐于使 M 相当大。但当 M 很大时，许多表都将是空的，而且 M 个表头的许多空间都将浪费掉。这就提示了另外一个当记录很小时可用的方法：我们可以把表头和记录的存储重叠起来，只提供 M 个记录和 M 个链接而不是 N 个记录和 $M+N$ 个链接所需的空间。有时有可能对所有数据作一趟扫描，找出哪一些表头将被使用，然后作另一趟扫描把所有的“溢出”记录都插入到空的位置上。但是这往往是不实际的或不可能的，我们宁可用一种每个记录只处理一次，即只在它第一次进入系统时处理一次的技术。下面由 F.A.Williams 给出的算法 [*CACM* **2**,**6** (1959 年 6 月)，21～24]是解决这个问题的一种方便的方法。

算法 C（拉链散列表的查找和插入）　本算法在一个 M 节点的表中查找一个给定的键码 K。如果 K 不在表中，且这个表不满，则插入 K。

表的诸节点，以 TABLE[i]来表示，$0 \leqslant i \leqslant M$，而且它们有两种不同的类型，即空的和已占用的。一个已占用的节点含有一个键码字段 KEY[i]，一个链接字段 LINK[i]，以及可能还有其它的字段。

这个算法利用了一个散列函数 $h(K)$，还使用一个辅助变量 R 来帮助寻找可用空间。当表空时，$R = M + 1$，而当进行插入时，对于在 $R \leqslant j \leqslant M$ 范围中的所有 j，TABLE[j]是已占用的。约定 TABLE[0]总是空的。

C1. [散列] 置 $i \leftarrow h(K) + 1$。(现在 $1 \leqslant i \leqslant M$)。

C2. [有一个表吗?] 如果 TABLE[i]是空的，则转到 C6。(否则 TABLE[i]已占用；我们将考察在这里开始的已占用节点的表列)。

C3. [比较] 如果 $K =$ KEY[i]，则这个算法成功地结束。

C4. [进到下一个] 如果 LINK[i]$\neq 0$，则置 $i \leftarrow$ LINK[i]并转回到步骤 C3。

C5. [找空节点] (这次查找不成功，要找表中的一个空位置。)R 减值一次或多次直到找到使得 TABLE[R]为空的 R 值为止。如果 $R = 0$，则这个算法以溢出(没有空节点)结束；否则置 LINK[i]$\leftarrow R$，$i \leftarrow R$。

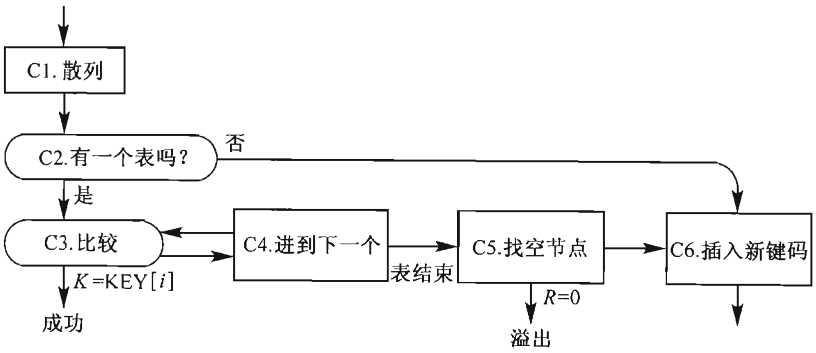


图 39 拉链散列表的查找和插入

C6. [插入新键码] 标记 TABLE[i]为一个已占用的节点，并作 KEY[i]$\leftarrow K$ 和 LINK[i]$\leftarrow 0$。 ▌

这个算法允许若干表结合起来，使得在诸记录插入到表中之后，就不必移动了。例如，见图 40，其中 SEKS 出现在含有 TO 和 FIRE 的表中，因为后者已经插入到位置 9 上。

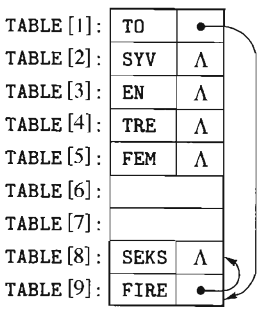


图 40 结合起来的拉链

为了看看这个算法同这一章的其它算法比较时情况如何，我们可以写出下列 MIX 程序。以下分析指出，已占用单元的表一般很短，而且程序设计已经考虑到这一点。

程序 C(拉链散列表的查找和插入) 为方便起见，

假定键码仅有三个字节长，且节点被表示如下：

空节点

−	1	0	0	0	0

；

已占用节点

+	LINK	KEY

．　　　　(13)

假定表的大小 M 是质数；TABLE[i]被存储在 TABLE + i 之中，rI1≡i，rA≡K；rI2≡LINK[i]和/或 R。

```
KEY   EQU   3:5
LINK  EQU   0:2
START LDX   K               1         C1. 散列
      ENTA  0               1
      DIV   =M=             1
      STX   *+1(0:2)        1
      ENT1  *               1         i←h(K)
      INC1  1               1         +1
      LDA   K               1
      LD2   TABLE,1(LINK)   1         C2. 有一个表否？
      J2N   6F              1         如果 TABLE[i]为空则转到 C6
      CMPA  TABLE,1(KEY)    A         C3. 比较
      JE    SUCCESS         A         如果 K = KEY[i]，则转出
      J2Z   5F              A − S1    如果 LINK[i] = 0，则转到 C5
4H    ENT1  0,2             C − 1     C4. 进到下一个
      CMPA  TABLE,1(KEY)    C − 1     C3. 比较
      JE    SUCCESS         C − 1     如果 K = KEY[i]则转出
      LD2   TABLE,1(LINK)   C − 1 − S2
      J2NZ  4B              C − 1 − S2 如果 LINK[i]≠0，则前进
5H    LD2   R               A − S     C5. 找空节点
      DEC2  1               T         R←S←1
      LDX   TABLE,2         T
      JXNN  *-2             T         重复直到 TABLE[R]为空
      J2Z   OVERFLOW        A − S     如果未剩下空节点则转出
      ST2   TABLE,1(LINK)   A − S     LINK[i]←R
      ENT1  0,2             A − S     i←R
```

```
        ST2   R                A - S    修改内存中的 R
6H      STZ   TABLE,1(LINK)    1 - S    C6. 插入新的键码
        STA   TASBLE,1(KEY)    1 - S    KEY[i]←K  ▌
```

这个程序的运行时间依赖于

C = 查找时所探查的表的项数

A = [初始探查发现一个已占用的节点]

S = [查找成功]

T = 寻求可用空间时所探查的表项数

这里 $S = S1 + S2$,其中如果第一次试验成功则 $S1 = 1$。程序 C 的查找阶段总共的运行时间是 $(7C + 2A + 17 - 3S + 2S1)u$,而当 $S = 0$ 时一个新键码的插入还需花费附加的 $(8A + 4T + 4)u$。

假设这个程序开始时表中有 N 个键码,而且设

$$\alpha = N/M = \text{这个表的负载因子} \tag{14}$$

如果散列函数是随机的,则在一次不成功的查找中 A 的平均值显然是 α,而且习题 39 证明在一次不成功的查找中 C 的平均值是

$$C'_N = 1 + \frac{1}{4}\left(\left(1 + \frac{2}{M}\right)^N - 1 - \frac{2N}{M}\right) \approx 1 + \frac{e^{2\alpha} - 1 - 2\alpha}{4} \tag{15}$$

于是,当这个表为半满时,在一次不成功的查找当中所作探查的平均次数大约是 $\frac{1}{4}(e+2) \approx 1.18$;甚至当这个表快要完全满时,在插入最后的项之前所作探查的平均次数,将仅仅大约是 $\frac{1}{4}(e^2+1) \approx 2.10$。如同在习题 40 中所证明的那样,标准差也小。这些统计证明,当散列函数是随机的时,即使算法偶尔允许诸表结合起来,这些表仍很短。如果散列函数是坏的或我们极不走运,当然 C 可以像 N 那样大。

在一次成功的查找中,我们总有 A = 1。在一次成功的查找期间探查的平均次数,可以计算如下:若假定每个键码都同等可能,则求 C + A 对前 N 次不成功的查找之和并除以 N。于是我们得到

$$C_N = \frac{1}{N}\sum_{0 \le k < N}\left(C'_k + \frac{k}{M}\right) = 1 + \frac{1}{8}\frac{M}{N}\left(\left(1 + \frac{2}{M}\right)^N - 1 - \frac{2N}{M}\right) + \frac{1}{4}\frac{N-1}{M} \approx 1 + \frac{e^{2\alpha} - 1 - 2\alpha}{8\alpha} + \frac{\alpha}{4} \tag{16}$$

作为在一次随机的成功查找中探查的平均次数。甚至从一个满的表中找出一个项,平均也才仅仅需要 1.80 次探查!类似地(见习题 42),S1 的平均值是

$$S1_N = 1 - \frac{1}{2}((N-1)/M) \approx 1 - \frac{1}{2}\alpha \tag{17}$$

乍一看去，步骤 $C5$ 可能显得很低效，因为它要顺序地查找一个空位置。但是实际上当一个表正被构造时，在步骤 $C5$ 中所作的表探查的总数决不超过表中的项目数；所以我们每作一次插入平均至多需要一次探查。习题 41 证明，在一次随机的不成功的查找中 T 近似于 αe^{α}。

有可能对算法 C 进行修改，使得任何两个表都不结合在一起，但是这样一来记录就要搬家了。例如，考虑在我们刚要把 SEKS 插入到位置 9 之前，图 40 的状况；为了保持表是分开的，必须移动 FIRE，而且为了这一目的，还要发现哪个节点是指向 FIRE 的。如同 D.E.Ferguson 所建议的，通过散列 FIRE 和往下查找它的表，我们无需提供两路链接就能解决这个问题，因为这些表都是短的；习题 34 证明，当诸表不相结合时，平均的探查次数，减少成

$$C'_N = 1 + \frac{N(N-1)}{2M^2} \approx 1 + \frac{\alpha^2}{2} \quad \text{（不成功的查找）} \tag{18}$$

$$C_N = 1 + \frac{N-1}{2M} \approx 1 + \frac{\alpha}{2} \quad \text{（成功的查找）} \tag{19}$$

对于(15)和(16)这个改进尚不足以构成改变这个算法的理由。

另一方面，Butler Lampson 已经发现，如果我们避免把这些表结合在一起，则在使用拉链方法时，大多数用于链接的空间都可以节省。这导致了在习题 13 中讨论的一个有趣的算法。Lampson 的方法在每一项中引入一个标志位。因此使得在一次不成功的查找中所需平均探查数稍微减少，从(18)减少成为

$$\left(1 - \frac{1}{M}\right)^N + \frac{N}{M} \approx e^{-\alpha} + \alpha \tag{18'}$$

当 $N > M$ 时，可以使用分开的拉链，所以在这种情况下溢出不是一个严重的问题。当诸表如同在图 40 中和算法 C 中那样结合起来时，我们可以把额外的一些项目链接成一个辅助的存储池：L.Guibas 已经证明，为插入第 $(M+L+1)$ 项需作的探查平均次数为 $\left(L/2M + \frac{1}{4}\right)((1+2/M)^M - 1) + 1/2$。然而，通常喜欢使用把头一个冲突的元素放进一个辅助存储区域的另一方案，并且仅当这个辅助区域已经满时才允许诸表结合在一起；见习题 43。

以“开式寻址法”解决冲突　解决冲突的另一个方法，是完全撤销链，简单地逐个考察表的各项，直到或者发现键码 K 或者发现一个空的位置。这个思想就是确定某种规则，通过它，每个键码 K 确定一个“探查序列”，即表中的一系列位置，每当要插入或查出 K 时这些位置即被探查。如果我们使用由 K 确定的探查序列，在查找 K 时，遇到一个空缺未用的位置，则可以得出结论，K 不在表中，因为在每次处理 K 时用的是同一个探查序列。此类通用的方法被 W.W.Peterson 称作开式寻址法 [*IBM J.Research & Development* **1**(1957)，130～146]。

称为线性探查的最简单的开式寻址方案,使用如同下列算法中那样的循环探查序列

$$h(K), h(K)-1, \cdots, 0, M-1, M-2, \cdots, h(K)+1 \tag{20}$$

算法 L(线性探查和插入) 这个算法在有 M 个节点的表中寻找一个给定的键码 K。如果 K 不在表中且表不满,则插入 K。

对于 $0 \leqslant i < M$,以 TABLE[i],表示表的节点,它们有两种不同的类型,即空的和已占用的。一个已占用的节点包含称为 KEY[i]的一个键码,可能还有其它的字段。一个辅助变量 N 用以记住占用了多少个节点;把 N 看做是表的一部分,而且当插入一个新键码时,N 的值增 1。

本算法利用一个散列函数 $h(K)$,并使用线性探查序列(20)查表。对该序列的修改将在下面讨论。

L1.[散列] 置 $i \leftarrow h(K)$。(现在 $0 \leqslant i < M$。)

L2.[比较] 如果 TABLE[i]为空,则转到 L4。否则如果 KEY[i] $= K$,则这个算法成功地结束。

L3.[进到下一个] 置 $i \leftarrow i-1$;如果现在 $i < 0$,则置 $i \leftarrow i + M$。转回到步骤 L2。

L4.[插入](查找是不成功的)如果 $N = M-1$,则算法以溢出结束。(这个算法认为当 $N = M-1$ 时,而不是当 $N = M$ 时表是满的,见习题 15)。否则置 $N \leftarrow N+1$,标记 TABLE[i]为已占用的,且置 KEY[i] $\leftarrow K$。 ▌

图 41 示出,当分别使用散列码 2,7,1,8,2,8,1,通过算法 L 插入七个示例性键码(11)时发生的情况;最后三个键码,FEM、SEKS 和 SYV,已经从它们开始的位置 $h(K)$被转移了。

0	FEM
1	TRE
2	EN
3	
4	
5	SYV
6	SEKS
7	TO
8	FIRE

图 41 线性开式寻址法

程序 L(线性探查和插入) 这个程序处理全字长的键码;但不允许值为 0 的键码,因为 0 被用作标志表中的一个空位置。(或者,我们可以要求键码是非负的,让空位含 -1)。假定表的大小 M 是素数,而且对于 $0 \leqslant i < M$ 把 TABLE[i]存在单元

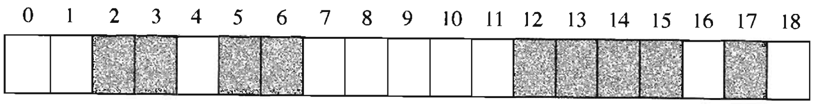
TABLE + i 中。为了加快内循环，假定单元 TABLE − 1 包含 0。假定单元 VACANCIES 包含值 $M-1-N$；且 $\mathrm{rA}\equiv K$，$\mathrm{rI1}\equiv i$。

为了加速这个程序的内循环，已经从循环中撤销了测试“$i<0$”，而仅仅保留了步骤 L2 和 L3 的必要部分。查找阶段的总共运行时间为 $(7C+9E+21-4S)u$，而在不成功查找之后的插入额外增加 $8u$。

```
START LDX   K            1          L1. 散列
      ENTA  0            1
      DIV   =M=          1
      STX   *+1(0:2)     1
      ENT1  *            1          i←h(K)
      LDA   K            1
      JMP   2F           1
8H    INC1  M+1          E          L3. 进行到下一个
3H    DEC1  1            C+E-1      i←i-1
2H    CMPA  TABLE,1      C+E        L2. 比较
      JE    SUCCESS      C+E        如果 K=KEY[i]则转出
      LDX   TABLE,1      C+E-S
      JXNZ  3B           C+E-S      如果 TABLE[i]非空，则转 L3
      JIN   8B           E+1-S      如果 i=-1，则以 i←M 转 L3
4H    LDX   VACANCIES    1-S        L4. 插入
      JXZ   OVERFLOW     1-S        如果 N=M-1，则溢出，并转出
      DECX  1            1-S
      STX   VACANCIES    1-S        N 增加 1
      STA   TABLE,1      1-S        TABLE[i]←K  ▌
```

如同在程序 C 中那样，变量 C 表示探查的次数，S 指出这次查找是否成功。我们可以忽略变量 E，它仅当对 TABLE[−1]已经作了一次假的探查时为 1，因为它的平均值是 $(C-1)/M$。

线性探查的经验表明，在表开始变满之前，本算法工作得十分好；但是这个过程终于减慢下来，而且长时间拖拉的查找逐渐地频繁起来。这只要考察 $M=19$ 和 $N=9$ 的假设的散列表，便可以明白这一特性的原因：



(21)

磁带阴影的方格表示已占用的位置。有待插入到表中的下一个键码 K 将进入到十个空的空间之一，但是这些空位并不是同等可能的；事实上，如果 $11\leqslant h(K)\leqslant$

15,K 将被插入到位置 11 中,同时仅当 $h(K)=8$ 时它才进入到位置 8。因此位置 11 的可能性,5 倍于位置 8;长的表列倾向于变得更长。

这个现象本身不足以看成是线性探查的一个相对拙劣的特性,因为在算法 C 中也出现类似的情况。(一个长度为 4 的表列在算法 C 下,其增长的可能性是长度为 1 的表列的 4 倍)。当在(21)中像 4 或 16 这样的单元被填入内容时,真正的问题出现了;于是两个分开的表列组合起来,而在算法 C 中的表列一次决不增长一步以上。因而,当 N 趋于 M 时,线性探查的性能迅速退化。

我们将在这一节的稍后证明,算法 L 所需要的平均探查次数近似于

$$C'_N \approx \frac{1}{2}\left(1+\left(\frac{1}{1-\alpha}\right)^2\right) \qquad \text{(不成功的查找)} \tag{22}$$

$$C_N \approx \frac{1}{2}\left(1+\frac{1}{1-\alpha}\right) \qquad \text{(成功的查找)} \tag{23}$$

其中 $\alpha=N/M$ 是该表的负载因子。所以当表的满载程度低于 75% 时,即使程序 C 处理的是不切实际地短的键码,程序 L 仍然几乎和程序 C 一样快。另一方面,当 α 趋于 1 时,对于程序 L 我们可以作的最好的评价是,它缓慢但稳当地运行着。事实上,当 $N=M-1$ 时,在表中仅有一个空位置,所以在一次不成功的查找中平均的探查次数是 $(M+1)/2$;我们也将证明当表满时,在一次成功的查找中平均的探查次数近似于 $\sqrt{\pi M/8}$。

当表快满时,堆积现象使线性探查变得代价高昂,而如果连续的键码值 $\{K,K+1,K+2,\cdots\}$ 出现的可能性很大,则使用除法散列更加剧了这种堆积现象,因为这些键码将有连续的散列码。乘法散列将令人满意地破坏这种堆积现象。

防止连续散列码问题的另一个方法是在步骤 L3 中置 $i \leftarrow i-c$,而不是 $i \leftarrow i-1$。c 为任何正值都行,只要它与 M *互素*,因为在这种情况下,探查序列仍将考察表中每一个位置。由于对于 $i<0$ 的测试,这样一种变化将使得程序 L 稍微慢些。减 c 而不是减 1 并不改变堆积现象,因为距离为 c 的记录组集仍将形成;等式(22)和(23)仍适用,但是连续的键码 $\{K,K+1,K+2,\cdots\}$ 的出现现在实际上是一种帮助,而不是一个障碍。

尽管 c 的一个固定值并不减少堆积现象,但通过命 c 依赖于 K,我们可以很好地改进这一情况!这一思想导致了对算法 L 的一项重要的修改,这是由 Guy de Balbine 首先发现的[*Ph.D. thesis, Calif. Inst. of Technology* (1968),149~150]。

算法 D(*使用两次散列的开式寻址法*) 这个算法和算法 L 几乎一样,但它以稍微不同的方式,通过使用两个散列函数 $h_1(K)$ 和 $h_2(K)$ 来探查表。和通常一样,$h_1(K)$ 产生 0 与 $M-1$ 之间(含 0 和 $M-1$)的一个值;但 $h_2(K)$ 必须产生与 M *互素*的 1 与 $M-1$ 之间的一个值(例如,如果 M 是素数,则 $h_2(K)$ 可以是 1 和 $M-1$ 之间的任何值(包括 1 和 $M-1$ 在内);或者如果 $M=2^m$,则 $h_2(K)$ 可以是 1 和 2^m-1 之间的任何奇数值)。

D1.[第一次散列]　置 $i \leftarrow h_1(K)$。

D2.[第一次探查]　如果 TABLE[i]是空的,则转到 D6。否则,如果 KEY[i] = K,则算法成功地结束。

D3.[第二次散列]　置 $c \leftarrow h_2(K)$。

D4.[前进到下一个]　置 $i \leftarrow i - c$;如果现在 $i<0$,则置 $i \leftarrow i + M$。

D5.[比较]　如果 TABLE[i]是空的,则转到 D6。否则如果 KEY[i] = K,则这个算法成功地结束。否则转回 D4。

D6.[插入]　如果 $N = M - 1$,则这个算法以溢出结束。否则置 $N \leftarrow N + 1$,标志 TABLE[i]已占用,且置 KEY[i] $\leftarrow K$。▌

关于计算 $h_2(K)$的若干可能性已被提出。如果 M 是素数,且 $h_1(K) = K \bmod M$,则我们可以设 $h_2(K) = 1 + (K \bmod (M-1))$;但由于 $M-1$ 是偶数,因而命 $h_2(K) = 1 + (K \bmod (M-2))$将更好。这提示选择 M 使得 M 和 $M-2$ 好像 1021 和 1019 那样是"孪生素数"。另外,我们可以置 $h_2(K) = 1 + (\lfloor K/M \rfloor \bmod (M-2))$,因为商$\lfloor K/M \rfloor$作为计算 $h_1(K)$的副产品,可在一个寄存器中得到。

如果 $M = 2^m$,且我们正在使用乘法散列,则只需左移 m 个以上的二进位和"或"上一个 1,即可计算出 $h_2(K)$,于是代码序列(5)之后应接上

```
ENTA 0      清 rA
SLB m       rAX 左移 m 位                (24)
OR =1=      rA ← rA ∨ 1
```

这要比除法方法更快些。

在上面提议的每一项技术当中,在下述意义下 $h_1(K)$和 $h_2(K)$是"独立的",即不同的键码对于 h_1 和 h_2 将有相同的值的概率是 $1/M^2$ 而不是 $1/M$。经验测试表明,具有独立的散列函数的算法 D 的特性,基本上也就同当键码随机地插入到表中时所需要的探查次数无异;实际上没有如同算法 L 中那样的"拥挤"或"堆积"。

如同 Gary Knott 于 1968 年所建议的那样,也有可能让 $h_2(K)$依赖于 $h_1(K)$;例如,如果 M 是素数,则我们可以命

$$h_2(K) = \begin{cases} 1 & \text{如果 } h_1(K) = 0 \\ M - h_1(K) & \text{如果 } h_1(K) > 0 \end{cases} \tag{25}$$

这将比作另一个除法更快,但是我们将看到,它却引起一定数量的*二次堆积*,因为增加了两个或更多个键码走同一条通路的机会,因此需要稍多些探查。以下导出的公式,可用来确定在散列时所获得的好处,是否超过探查时间上的损失。

算法 L 和 D 是非常类似的,但是也还有一定的差别,因此比较它们相应的 MIX 程序的运行时间是有益的。

程序 D(使用两次散列的开式寻址法)　由于这个程序大体上很类似于程序 L,因此就不加注释。rI2≡$c-1$。

```
START LDX    K               1
      ENTA   0               1
      DIV    =M=             1
      STX    *+1(0:2)        1
      ENT1   *               1
      LDX    TABLE,1         1
      CMPX   K               1
      JE     SUCCESS         1
      JXZ    4F              1-S1
      SRAX   5               A-S1
      DIV    =M-2=           A-S1
      STX    *+1(0:2)        A-S1
      ENT2   *               A-S1
      LDA    K               A-S1
3H    DEC1   1,2             C-1
      J1NN   *+2             C-1
      INC1   M               B
      CMPA   TABLE,1         C-1
      JE     SUCCESS         C-1
      LDX    TABLE,1         C-1-S2
      JXNZ   3B              C-1-S2
4H    LDX    VACANCIES       1-S
      JXZ    OVERFLOW        1-S
      DECX   1               1-S
      STX    VACANCIES       1-S
      LDA    K               1-S
      STA    TABLE,1         1-S ▌
```

在这个程序中的频率计数 $A,C,S1,S2$,有与上面程序 C 中类似的解释。另一个变量 B 平均将是$(C-1)/2$。(如果我们限制 $h_2(K)$ 的范围为,比如说,$1\leqslant h_2(K)\leqslant M/2$,则 B 大约将仅仅是$(C-1)/4$;速度的这一提高大约将不足于弥补在探查次数方面的显著增加。)当在这个表中有 $N=\alpha M$ 个键码时,在一次不成功的查找中,A 的平均值当然是 α,而在一次成功的查找中 $A=1$。如同在算法 C 中那样,在一次成功的查找中 $S1$ 的平均值是 $1-\frac{1}{2}((N-1)/M)\approx 1-\frac{1}{2}\alpha$。探查的平均次数难以精确地确定,但经验测试证明以下导出的公式同“一致探查”相当符合,即当 $h_1(K)$ 和 $h_2(K)$ 独立时,有

$$C'_N=\frac{M+1}{M+1-N}\approx(1-\alpha)^{-1}\qquad(\text{不成功的查找})\tag{26}$$

$$C_N=\frac{M+1}{N}(H_{M+1}-H_{M+1-N})\approx-\alpha^{-1}\ln(1-\alpha)\qquad(\text{成功的查找})\tag{27}$$

当 $h_2(K)$ 如同在(25)中那样依赖于 $h_1(K)$ 时,二次堆积使这些公式增加到

$$C'_N=\frac{M+1}{M+1-N}-\frac{N}{M+1}+H_{M+1}-H_{M+1-N}+O(M^{-1})\approx(1-\alpha)^{-1}-\alpha-\ln(1-\alpha)\tag{28}$$

$$C_N=1+H_{M+1}-H_{M+1-N}-\frac{N}{2(M+1)}-(H_{M+1}-H_{M+1-N})/N+O(N^{-1})\approx 1-\ln(1-\alpha)-\frac{1}{2}\alpha\tag{29}$$

(见习题 44)。注意,随着表变满,即当 $N = M$ 时,C_N 的这些值分别趋向于 $H_{M+1} - 1$ 和 $H_{M+1} - \frac{1}{2}$;这比我们在算法 L 中所发现的更好些,但不如拉链方法那样好。

由于每次探查花费的时间比算法 L 稍少,故双散列仅当表变满时才有利。图 42 比较了程序 L、程序 D 以及修改后的程序 D 的平均运行时间,修改后的程序 D 包含二次堆积,并以下列三条指令

```
ENN2  1 - M,1   c ← M - i
J1NZ   * + 2                                (30)
ENT2  0         如果 i = 0 则 c ← 1
```

来代替行 10～13 中 $h_2(K)$的相当慢的计算。程序 D 总共花费 $8C + 19A + B + 26 - 13S - 17S1$ 时间单位;在一次成功的查找中(30)的修改大约节省这些时间单位中的 $15(A - S1) \approx 7.5\alpha$。在这种情况下,二次堆积比独立的两次散列更为可取。

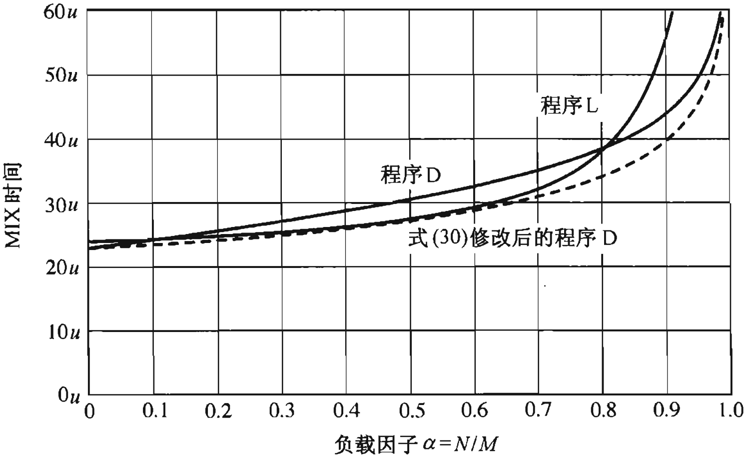


图 42　通过三个开式寻址方案进行成功查找的运行时间

在一台二进计算机上,我们可以以另一种方式来加速 $h_2(K)$的计算,比如说如果 M 是一个大于 512 的素数,则以

```
AND    =511=          rA ← rA mod 512
STA    * + 1 (0 : 2)                        (31)
ENT2   *              c ← rA + 1
```

来代替行 10～13。这一思想(由 Belll 和 Kaman 提出,*CACM* **13**(1970),675～677,他们独立地发现了算法 D)无需用任何除法就避免了二次堆积。

已经提出许多其它的探查序列来改进算法 L,但看来没有一个比算法 D 优越,可能习题 20 中描述的方法除外。

通过使用诸键码的相对次序,我们可以把算法 L 或 D 的不成功查找的平均运行时间减少到成功查找的平均运行时间;参见习题 66。这个技术在不成功查找很

普遍的那些应用中很重要;例如,当 TEX 查找对于它的破折号规则的例外情况时,就使用这样一个算法。

Brent 的变形 Richard P. Brent 已经发现了修改算法 D 的一个方法,使得当表变满时平均的成功查找时间保持有界。他的方法[*CACM* **16** (1973),105～109]是以这样一个事实为基础的,即在许多应用中,成功的查找比插入要频繁得多;因此,他建议在插入一项目时,多做一些工作,包括移动表中的记录,以便减少预期的查找时间。

例如,图 43 示出了在一个典型的 PL/I 过程中,每个标识符真正被查到的次数。这一思想指出,使用一个散列表记住变量名字的 PL/I 编译程序,将查找许多名字五次以上,但插入只一次。类似地,Bell 和 Kaman 发现,一个 COBOL 编译程序在编译一个程序时,使用它的符号表算法 10988 次,但只对该表作了 735 次插入;就是每作一次不成功的查找平均大约要作 14 次成功的查找。有时一个表实际上仅被建立一次(例如,在一个汇编程序中的符号操作码表),而此后纯粹用作检索。

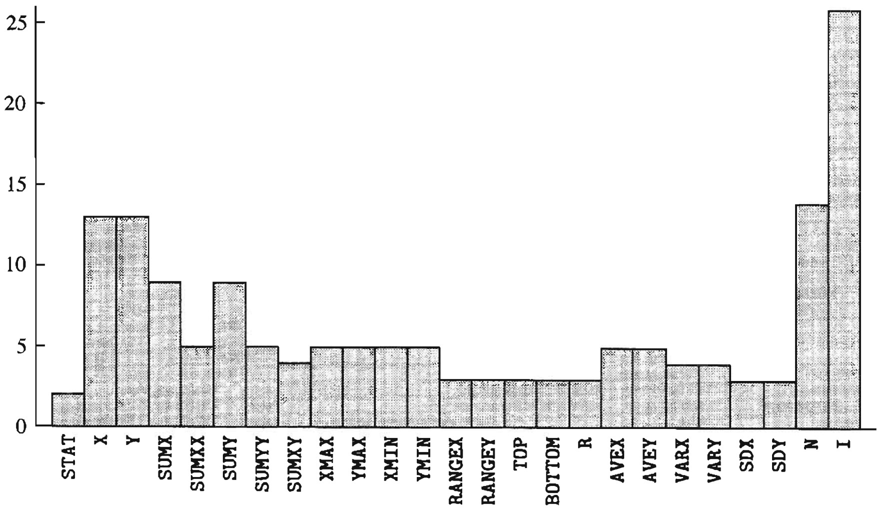


图 43 一个编译程序典型地查找变量名的次数,诸名字按照它们头一次出现的次序自左到右地列出

Brent 的思想是把算法 D 中的插入过程改变如下。假设一次不成功的查找要探查单元 $p_0, p_1, \cdots p_{t-1}, p_t$,其中 $p_j = (h_1(K) - jh_2(K)) \bmod M$,且 TABLE[$p_t$]为空。如果 $t \leqslant 1$,则我们像通常那样把 K 插入到位置 p_t 中;但如果 $t \geqslant 2$,则计算 $c_0 = h_2(K_0)$,其中 $K_0 =$ KEY[p_0],而且看 TABLE[$(p_0 - c_0) \bmod M$]是否为空。如果为空,则置它为 TABLE[p_0],而后把 K 插入到位置 p_0 中,这增多了一步对于 K_0 的查找时间,但它减少了 $t \geqslant 2$ 步对于 K 的查找时间,所以结果还是改进了。类似地,如果 TABLE[$(p_0 - c_0) \bmod M$]是已占用的,而且 $t \geqslant 3$,则我们尝试 TABLE[$(p_0 - 2c_0) \bmod M$];

如果这也是满的，则计算 $c_1=h_2(\text{KEY}[p_1])$和试验 TABLE$[(p_1-c_1) \bmod M]$；等等。一般地说，命 $c_j=h_2(\text{KEY}[p_j])$且 $p_{j,k}=(p_j-kc_j) \bmod M$；如果我们发现 TABLE$[p_{j,k}]$对于使得 $j+k<r$ 的所有下标 j 和 k 是已占用的，而且如果 $t\geqslant r+1$，则考察 TABLE$[p_{0,r}]$，TABLE$[p_{1,r-1}]$，…，TABLE$[p_{r-1,1}]$。如果头一个空位出现于位置 $p_{j,r-j}$处，则我们就置 TABLE$[p_{j,r-j}]\leftarrow$TABLE$[p_j]$，而且把 K 插入位置 p_j 中。

Brent 的分析指出，正如后面的图 44 中所示的那样，每次成功的查找的平均探查数是减少了，极大值大约为 2.49。

Brent 的变形并不减少在不成功查找中探查的次数 $t+1$；它保持在等式(26)所指出的水平，当表满时，趋于$\frac{1}{2}(M+1)$。每次插入需要计算的平均次数 h_2 是 $\alpha^2+\alpha^5+\frac{1}{3}\alpha^6+\cdots$，按照 Brent 的分析，最终趋于 $\Theta(\sqrt{M})$；而且在判断如何来进行插入时额外被探查的表中位置数大约是 $\alpha^2+\alpha^4+\frac{4}{3}\alpha^5+\alpha^6+\cdots$。

E. G. Mallach[*Comp. J.* **20** (1977)，137～140]已经以 Brent 的变形的改进作了实验，而 Gaston H. Gonnet 和 J. Ian Munro 已经得到了一些进一步的结果[*SICOMP* **8** (1979)，463～478]。

删去　许多计算机程序员都高度地信任算法，他们惊奇地发现从散列表删去记录的一种显然的方式居然失效。例如，如果我们想从图 41 删去键码 EN，则不能简单地把表中这个位置标成空的，因为这将使另一个键码 FEM 突然地被忘记！（回想一下，EN 和 FEM 两者是散列到同一单元的。当考察 FEM 时，我们将发现一个空的位置，从而表示一次不成功的查找。）由于诸表的接合，对于算法 C 必出现类似的问题；试想像从图 40 中删去 TO 和 FIRE 两者。

一般地说，我们可以通过置一个特殊的代码值于对应的单元中来处理删去，使得有三类表的项：空的、已占用的以及已删去的。当查找一个键码时，我们将跳过已删去的单元，就像它们是已占用的那样。如果查找是不成功的，则这个键码可被插入到头一个遇到的删去的或者空的位置中。

但是这一思想仅当删去非常稀少时才是可行的，这是因为一旦表的项变为已占用的，则它们就决不再次地变空。在进行了一长串重复的插入到删去的序列之后，所有可用空间都终将消失，而且每次不成功的查找将花费 M 个探查！其次每个探查的时间将增加，因为我们要测试 i 是否已经恢复成它在步骤 D4 中的开始值；而且在一次成功的查找中探查的次数将趋于从 C_N 上升到 C'_N。

当正在使用的是线性探查（即算法 L）时，如果我们愿意对删去做某些额外工作，则在删去时可避免上述麻烦情况。

算法 R（用线性探查实现删去）　假定通过算法 L 已经构造了一个开式散列表，本算法从一个给定的位置 TABLE$[i]$删去记录。

R1.[弄空一个单元]　标记 TABLE$[i]$为空，且置 $j\leftarrow i$。

R2. [i 减值] 置 $i \leftarrow i-1$,如果这使得 i 成为负的,则置 $i \leftarrow i+M$。

R3. [检查 TABLE[i]] 如果 TABLE[i]是空的,则这算法结束。否则置 $r \leftarrow h$(KEY[i]),这个键码原来的散列地址现在存在位置 i 中。如果 $i \leqslant r < j$ 或如果 $r < j < i$ 或 $j < i \leqslant r$(换言之,如果 r 循环地位于 i 和 j 之间),则转回到 $R2$。

R4. [移动一个记录] 置 TABLE[j]←TABLE[i]并返回步骤 $R1$。 ▌

习题 22 表明这个算法不引起性能上的蜕化;换言之,在等式(22)和(23)中预测的平均探查次数保持相同(定理 6.2.2H 中证明了对于树插入的一个较弱的结果)。但是算法 R 的正确性与这样一个事实密切相关,即它是用于线性探查的,而且不可能有类似的和算法 D 一起使用的删去过程。习题 64 中分析了算法 R 的平均运行时间。

当然,当对每个可能的散列值使用分开的表并实行拉链时,删去不会引起问题,因为它仅仅是从一个链接的线性表的删去。习题 23 中讨论了算法 C 中的删去。

算法 R 可能移动表中的某些项,但如果有其它某处是指向这些项的,则这种移动是不希望发生的。通过采用废料回收中使用的某些思想,有可能有另外一个删去的方法(参见 2.3.5 小节):对于每个键码我们可以保持一个"访问计数"说明有多少其它的键码同它接触;然后当某个未占用单元的访问计数为 0 时,就有可能把它转换成空状态。另一种方法是,每当已经累计有太多的删去的项时,我们可以扫视整个的表,同时转换所有未占用的位置成为空状态,而后考察所有剩余的键码,为的是找出哪些未占用的位置仍然需要"删去"状态。这些过程最初是由 T Gunji 和 E. Goto 提出的[*J Intformation Proc* **3** (1980),1～12],它避免使用浮动,且对任何散列技术都有效。

对算法的分析 了解一个散列方法的平均特性是特别重要的,因为每当进行散列时,我们只能依靠概率的法则。这些算法最坏的情况几乎都不可想像地坏,所以我们需要得到保证:平均特性是很好的。

在进行对线性探查等等的分析之前,让我们考虑这一状况的非常近似的模型,即所谓的*均匀的散列*。在这个由 W. W. Peterson[*IBM J. Research & Development* **1** (1957)]提出的模型中,我们假定每个键码被放置在这个表中完全随机的位置,使得 N 个已占用单元及 M-N 个空单元的 $\binom{M}{N}$ 种可能的配置是同等可能的。这个模型忽略一次堆积和二次堆积的任何影响;表中每个单元的占用基本上同其它的单元无关。对于这一模型,为插入第 $N+1$ 项恰恰需要 r 次探查的概率是 $r-1$ 个给定的单元已占用和另一个单元是空的这样的配置数,除以 $\binom{M}{N}$,即

$$P_r = \binom{M-r}{N-r+1} \Big/ \binom{M}{N}$$

因此,对于均匀的散列的平均探查次数是

$$C'_N=\sum_{r=1}^{M}rP_r=M+1-\sum_{r=1}^{M}(M+1-r)P_r=$$
$$M+1-\sum_{r=1}^{M}(M+1-r)\binom{M-r}{M-N-1}\bigg/\binom{M}{N}=$$
$$M+1-\sum_{r=1}^{M}(M-N)\binom{M+1-r}{M-N}\bigg/\binom{M}{N}=$$
$$M+1-(M-N)\binom{M+1}{M-N+1}\bigg/\binom{M}{N}=$$
$$M+1-(M-N)\frac{M+1}{M-N+1}=\frac{M+1}{M-N+1},\quad \text{对于 } 1\leqslant N<M \tag{32}$$

(在习题 3.4.2-5 中我们实质上曾解决了同随机抽样相联系的同一个问题)。置 $\alpha=N/M$, C'_N 的这一精确公式近似地等于

$$\frac{1}{1-\alpha}=1+\alpha+\alpha^2+\alpha^3+\cdots \tag{33}$$

这个级数有一种粗略的直观解释:我们需要一次以上的探查的概率为 α,二次以上的探查的概率为 α^2,等等。对于一次成功的查找对应的平均探查次数是

$$C_N=\frac{1}{N}\sum_{k=0}^{N-1}C'_k=\frac{M+1}{N}\left(\frac{1}{M+1}+\frac{1}{M}+\cdots+\frac{1}{M-N+2}\right)=$$
$$\frac{M+1}{N}(H_{M+1}-H_{M-N+1})\approx\frac{1}{\alpha}\ln\frac{1}{1-\alpha} \tag{34}$$

如同上边所说的,广泛的测试表明,在所有的实际应用中,具有两个独立的散列函数的算法 D 的性能实质上像均匀散列一样。事实上,在当 $M\to\infty$ 时的极限下,双散列渐进地等价于均匀探查(参见习题 70)。

这就完成了对均匀探查的分析。为了研究线性探查和其它类型的解决冲突的办法,我们需要以不同的更现实的方式来建立这个理论。我们为此目的采用的概率模型假定:M^N 个可能的"散列序列"的每一个

$$a_1a_2\cdots a_N \qquad 0\leqslant a_j<M \tag{35}$$

是同等可能的,其中 a_j 表示插入到表中的第 j 个键码的初始散列地址。给定任何特殊的查找方法,在一次成功的查找中探查的平均次数将如同上面那样以 C_N 表示;假定这是为求第 K 个键码所需要的探查的平均次数,平均是对所有 $1\leqslant k\leqslant N$ 取的,而且每个键码都是同等可能的。类似地,当第 N 个键码被插入时,所需要的探查的平均次数将以 C'_{N-1} 表示,其中认为所有的序列(35)是同等可能的;这是在以表中的 $N-1$ 个元素开始的不成功的查找中,探查的平均次数。当使用开式寻址法时

$$C_N=\frac{1}{N}\sum_{k=0}^{N-1}C'_k \tag{36}$$

于是我们可以像在(34)中所做的那样从其它量导出另一个量。

严格地说,即使在这个更为精确的模型中也有两个不足之处。首先,不同的散列序列不是全都同等可能的,因为诸键码本身是不同的。这使得 $a_1 = a_2$ 的概率稍微小于 $1/M$;但这个差别通常是可忽略的,因为所有可能的键码的集合相对于 M 来说是非常大的(见习题 24)。其次,一个好的散列函数将利用典型数据的非随机性,使它甚至不大能有 $a_1 = a_2$;结果,我们对于探查次数的估计将是悲观的。在这个模型中的另一点不精确性已在图 43 中指出:较早出现的键码(除某些例外者外),比后来出现的键码更可能被查找。因此我们对于 C_N 的估计更加悲观,而这些算法的实际运行情况应该比我们的分析所预言的稍好些。

在打过这些招呼之后,我们已经准备好了对于线性探查作一个"精确"的分析*。设 $f(M,N)$ 是在算法 L 插入了诸键码之后,使得表的位置 0 成为空的散列序列(35)的个数。线性探查的循环对称性意味着位置 0 为空的可能性恰如其它位置一样,所以它为空的概率是 $1 - N/M$;换言之,

$$f(M,N) = \left(1 - \frac{N}{M}\right)M^N \tag{37}$$

由约定,我们也置 $f(0,0)=1$。现在设 $g(M,N,k)$ 是使得这算法保持位置 0 为空、位置 1 到 k 是已占用的、位置 $k+1$ 为空的散列序列(35)的个数,我们有

$$g(M,N,k) = \binom{N}{k}f(k+1,k)f(M-k-1,N-k) \tag{38}$$

因为所有这样的散列序列都是由两个子序列组成的,一个序列(包含 k 个元素 $a_i \leqslant k$)保持位置 0 为空,并保持位置 1 到 k 是已占用的,另一个序列(包含 $N-k$ 个元素 $a_j \geqslant k+1$)保持位置 $k+1$ 为空;前一种类型有 $f(k+1,k)$ 个子序列,后一种类型有 $f(M-k-1,N-k)$ 个子序列,而且有 $\binom{N}{k}$ 种方式来散开两个这样的子序列。最后,设 P_k 是当插入第 $N+1$ 个键码时,恰巧需要 $k+1$ 个探查的概率;由此得出(见习题 25)

$$P_k = M^{-N}(g(M,N,k) + g(M,N,k+1) + \cdots + g(M,N,N)) \tag{39}$$

现在,$C'_N = \sum_{k=0}^{N}(k+1)P_k$;把这个等式同(36)~(39)放在一起并简化之,即得下列结果。

定理 K *假定所有 M^N 个散列序列(35)是同等可能的,则算法 L 所需要的平均探查次数是*

$$C_N = \frac{1}{2}(1 + Q_0(M,N-1)) \qquad (\text{成功的查找}) \tag{40}$$

$$C'_N = \frac{1}{2}(1 + Q_1(M,N)) \qquad (\text{不成功的查找}) \tag{41}$$

* 在这里作者禁不住想要插进一段关于历史的说明:我于 1962 年在开始撰写本书后不久,就已写出了下面的推导。由于这是我满意地分析过的头一个不平凡的算法,因此它对于这些书的结构已经有了强烈的影响。就是自那时起,算法分析实际上已成为我的生活的主旋律之一。——原注

其中

$$Q_r(M,N)=\binom{r}{0}+\binom{r+1}{1}\frac{N}{M}+\binom{r+2}{2}\frac{N(N-1)}{M^2}+\cdots=$$
$$\sum_{k\geqslant 0}\binom{r+k}{k}\frac{N}{M}\frac{N-1}{M}\cdots\frac{N-k+1}{M} \tag{42}$$

证明 计算的细节在习题 27 中给出。(关于方差,参见习题 28,67 和 68)。 ▌

在这个定理中出现的看起来稍微有些奇怪的函数 $Q_r(M,N)$,实际上并不难处理。我们有

$$N^k-\binom{k}{2}N^{k-1}\leqslant N(N-1)\cdots(N-k+1)\leqslant N^k$$

因此如果 $N/M=\alpha$,则

$$\sum_{k\geqslant 0}\binom{r+k}{k}\left(N^k-\binom{k}{2}N^{k-1}\right)/M^k\leqslant Q_r(M,N)\leqslant\sum_{k\geqslant 0}\binom{r+k}{k}N^k/M^k,$$
$$\sum_{k\geqslant 0}\binom{r+k}{k}\alpha^k-\frac{\alpha}{M}\sum_{k\geqslant 0}\binom{r+k}{k}\binom{k}{2}\alpha^{k-2}\leqslant Q_r(M,\alpha M)\leqslant\sum_{k\geqslant 0}\binom{r+k}{k}\alpha^k$$

即

$$\frac{1}{(1-\alpha)^{r+1}}-\frac{1}{M}\binom{r+2}{2}\frac{\alpha}{(1-\alpha)^{r+3}}\leqslant Q_r(M,\alpha M)\leqslant\frac{1}{(1-\alpha)^{r+1}} \tag{43}$$

当 M 很大,且 α 不太接近于 1 时,这个关系给了我们对于 $Q_r(M,N)$的一个好的估计(下限是比上限更好的近似)。当 α 趋于 1 时,这些公式就没有用了,但幸而 $Q_0(M,M-1)$是函数 $Q(M)$,它的渐近特性在 1.2.11.3 小节中已被详细地研究过了。而且,$Q_1(M,M-1)$就等于 M(见习题 50)。借助于超几何函数的标准记号,即等式 1.2.6-39,我们有

$$Q_r(M,N)=F(r+1,-N;;-1/M)=F\left(\begin{matrix}r+1,-N,1\\1\end{matrix}\middle|-\frac{1}{M}\right)$$

分析线性探查的另一个方法,已为 G. Schay, Jr 和 W. G. Spruth 得到[*CACM* **5** (1962),459～462]。尽管他们的方法仅产生对于定理 K 中的精确公式的一个近似,但它却对这个算法给出了进一步的分析,所以在这里我们将简单地概述一下。首先,让我们考虑由 W. W. Peterson 于 1957 年首先指出的线性探查的一个令人惊异的性质。

定理 P *通过算法 L 进行的一次成功的查找中,探查的平均次数同键码被插入的次序无关;它仅依赖于散列成每个地址的键码的数目。*

换言之,一个散列序列 $a_1a_2\cdots a_N$ 的任何重新排列,产生另一个散列序列,它的

诸键码相对于散列地址的平均偏移和原序列一样。(如同早先指出的,我们假定在这个表中的所有键码都有相同的重要性。如果某个键码比其它键码更频繁地被访问,则可以扩充本地证明,通过使用定理 6.1S 的方法,如果以频率的递减的次序插入它们,则出现一种最优的安排。)

证明 只需证明,对于散列序列 $a_1a_2\cdots aN$,为插入键码所需要的探查的总次数,和对于 $a_1\cdots a_{i-1}a_{i+1}a_ia_{i+2}\cdots a_N$,$1\leqslant i<N$ 所需要的总次数是相同的。显然,除开在第二个序列中的第 $i+1$ 个键码落入到在第一个序列中由第 i 个键码占据的位置外,两者没有差别。但第 i 个和第 $i+1$ 个仅仅交换了位置,所以对于第 $i+1$ 个的探查次数减少的数量,等同于对第 i 个的探查次数增加的数量。 ▌

定理 P 告诉我们,一个散列序列 $a_1a_2\cdots a_N$ 的平均查找长度,可由数 $b_0b_1\cdots b_{M-1}$ 确定,其中 b_j 是等于 j 的诸 a 的个数。从这个序列我们可以确定“进位序列” $c_0c_1\cdots c_{M-1}$,其中 c_j 是这样的一些键码的个数,在这些键码被插入时单元 j 和 $j-1$ 都被探查。这个序列通过规则

$$c_j=\begin{cases}0, & \text{如果 } b_j=c_{(j+1)\bmod M}=0\\ b_j+c_{(j+1)\bmod M}-1, & \text{否则}\end{cases}\tag{44}$$

确定。例如,设 $M=10$,$N=8$ 以及 $b_0\cdots b_9=0\ 3\ 2\ 0\ 1\ 0\ 0\ 0\ 0\ 2$;则 $c_0\cdots c_9=2\ 3\ 1\ 0\ 0\ 0\ 0\ 1\ 2\ 3$,因为一个键码需要从位置 2“进位”到位置 1,三个键码需要从位置 1 进到位置 0,其中的两个从位置 0 进到位置 9,等等。我们有 $b_0+b_1+\cdots+b_{M-1}=N$,而且为了查找 N 个键码所需要的平均探查次数是

$$1+(c_0+c_1+\cdots+c_{M-1})/N\tag{45}$$

规则(44)似乎是用 c 来定义自己的一种循环定义,但实际上每当 $N<M$ 时,所述的方程就有惟一的一个解(见习题 32)。

Schay 和 Spruth 使用了这一思想,借助于 $b_j=k$ 的概率 p_k,来确定 $c_j=k$ 的概率 q_k(这些概率同 j 无关)。于是

$$\begin{aligned}q_0&=p_0q_0+p_1q_0+p_0q_1\\ q_1&=p_2q_0+p_1q_1+p_0q_2\\ q_2&=p_3q_0+p_2q_1+p_1q_2+p_0q_3\end{aligned}\tag{46}$$

等等,因为例如,$c_j=2$ 的概率是 $b_j+c_{(j+1)\bmod M}=3$ 的概率。设 $B(z)=\Sigma p_kz^k$ 和 $C(z)=\Sigma q_kz^k$ 是这些概率分布的生成函数;等式(46)等价于

$$B(z)C(z)=p_0q_0+(q_0-p_0q_0)z+q_1z^2+\cdots=p_0q_0(1-z)+zC(z)$$

由于 $B(1)=1$,我们可以写 $B(z)=1+(z-1)D(z)$,而且由此得出

$$C(z)=\frac{p_0q_0}{1-D(z)}=\frac{1-D(1)}{1-D(z)}\tag{47}$$

因为 $C(1)=1$。根据(45),为进行查找所需要的平均探查次数,将是

$$1+\frac{M}{N}C'(1)=1+\frac{M}{N}\frac{D'(1)}{1-D(1)}=1+\frac{M}{2N}\frac{B''(1)}{1-B'(1)} \tag{48}$$

由于我们假定了每一个散列序列 $a_1\cdots a_N$ 是同等可能的，所以有

$$p_k=\Pr(\text{对于固定的 } j\text{，恰有 } k \text{ 个 } a_i \text{ 等于 } j)=\binom{N}{k}\left(\frac{1}{M}\right)^k\left(1-\frac{1}{M}\right)^{N-k} \tag{49}$$

因此

$$B(z)=1+\left(\frac{z-1}{M}\right)^N,\quad B'(1)=\frac{N}{M},\quad B''(1)=\frac{N(N-1)}{M^2} \tag{50}$$

而且根据(48)，平均探查次数将是

$$C_N=\frac{1}{2}\left(1+\frac{M-1}{M-N}\right) \tag{51}$$

读者能否看出，引起这答案不同于定理 K 中的正确结果的不正确推理(参见习题 33)。

***最优性考虑**　我们已经看到若干关于开式寻址法的探查序列的例子，因而自然要问：能否找到一个在某种意义下是最优的序列？这一问题已经由 J. D. Ullman 以下列有趣的方式提出来[*JACM* **19** (1972), 569～575]；我们不去计算一个散列地址 $h(K)$，而是把每个键码 K 都映射到 $\{0,1,\cdots,M-1\}$ 的一整个排列中，这排列表示对 K 使用的探查序列。对于 $M!$ 个排列的每一个赋予一个概率，而且提出一个广义的散列函数，它以指定的概率来选择相应的列。问题是："在其对应的平均探查次数 C_N 或 C'_N 为极小这样的意义下，对于诸排列选定什么样的概率，将给出最好的性能？"

例如，如果对每个排列选定 $1/M!$ 的概率，则容易看出我们恰巧有在(32)、(34)中已经分析过的均匀散列的特性。然而，Ullman 已经找到了对于 $M=4, N=2$ 的一个例子，对于它 C'_N 小于由均匀散列得到的值 $\frac{5}{3}$。他的构造是，对于除下列 6 个排列之外的所有排列，都指定概率为 0：

排列	概率	排列	概率
0 1 2 3	$(1+2\epsilon)/6$	1 0 3 2	$(1+2\epsilon)/6$
2 0 1 3	$(1-\epsilon)/6$	2 1 0 3	$(1-\epsilon)/6$
3 0 1 2	$(1-\epsilon)/6$	3 1 0 2	$(1-\epsilon)/6$

(52)

粗略地说，第一种探查趋向于 2 或 3，但第二种探查总是 0 或 1。为插入第三项所需要的平均探查次数 C'_3 是 $\frac{5}{3}-\frac{1}{9}\epsilon+O(\epsilon^2)$。所以我们通过取 ϵ 为一个小的正值，可以改进均匀散列。

然而，对于这些概率对应的 C'_1 的值是 $\frac{23}{18}+O(\epsilon)$，它大于 5/4(均匀散列的值)。Ullman 已经证明，对于某个 N，使得 $C'_N<(M+1)/(M+1-N)$ 成立的任何指定，

总是意味着对于某个 $n<N, C'_n>(M+1)/(M+1-n)$；你不可能在所有时候都胜过均匀散列。

实际上，一次成功的查找的探查次数 C_N，是比 C'_N 更好的量度。(52)中的排列对于任何的 N 都不能改进。而且 Ullman 确实猜测，对概率的任何选定，都不能使 C_N 小于均匀的值 $((M+1)/N)(H_{M+1}-H_{M+1-N})$。姚期智通过证明当 $N=\alpha M$ 和 $M\to\infty$ 时极限费用总是 $\geqslant\frac{1}{\alpha}\ln\frac{1}{1-\alpha}$，而证明了这个猜测的一个渐进形式[*JA*(*CM* **32** (1985),687～693)。

Ullman 猜测的强的形式似乎非常难以证明，特别是因为有许多方法可选定概率以达到均匀散列的效果；我们不需要对每个排列选定 $1/M!$。例如，对 $M=4$ 的下列选定就等价于均匀散列：

排列	概率	排列	概率
0 1 2 3	1/6	0 2 1 3	1/12
1 2 3 0	1/6	1 3 2 0	1/12
2 3 0 1	1/6	2 0 3 1	1/12
3 0 1 2	1/6	3 1 0 2	1/12

(53)

同时选定其它 16 个排列的概率为 0。

下列定理表征了产生均匀散列特性的所有选定。

定理 U *对于 $0<N<M$，在进行了 N 次插入之后，对于排列的一个概率选定将使 $\binom{M}{N}$ 个空的和已占用单元的配置的每一个都同等可能，当且仅当对所有的 N 和所有 N 元素的集合，选定所有如下排列的概率之和是 $1/\binom{M}{N}$，这些排列的头 N 个元素是一个给定的 N 元素集合的元素。*

例如，对于在某一顺序下以数 $\{0,1,2\}$ 开始的 $3!\,(M-3)!$ 个排列中的每个排列选定的概率之和必然是 $1/\binom{M}{3}=3!\,(M-3)!\,/M!$。注意，这个定理的条件在(53)中成立，因为 $1/L+1/12=1/4$。

证明 设 $A\subseteq\{0,1,\cdots,M-1\}$，且设 $\Pi(A)$ 是其头 $|A|$ 个元素为 A 的成员的所有排列之集合；且设 $S(A)$ 是对于这些排列选定的概率之和。设 $P_k(A)$ 是在执行开式寻址法过程中头 $|A|$ 次插入占用由 A 确定的位置，且最后一次插入恰需 k 次探查的概率；最后，设 $P(A)=P_1(A)+P_2(A)+\cdots$。本证明对 $N\geqslant1$ 用归纳法，假定对于满足 $|A|=n<N$ 的所有集合 A 有

$$P(A)=S(A)=1\Big/\binom{M}{n}$$

设 B 是任意 N 个元素的集合，则

$$P_k(B) = \sum_{\substack{A \subseteq B \\ |A| = k}} \sum_{\pi \in \Pi(A)} \Pr(\pi) P(B \setminus \{\pi_k\})$$

其中 $\Pr(\pi)$是对排列 π 选定的概率,而 π_k 是它的第k 个元素。由归纳法

$$P_k(B) = \sum_{\substack{A \subseteq B \\ |A| = K}} \frac{1}{\binom{M}{N-1}} \sum_{\pi \in \Pi(A)} \Pr(\pi),$$

它等于

$$\binom{N}{k} \Big/ \binom{M}{N-1}\binom{M}{k} \quad \text{如果 } k < N$$

因此

$$P(B) = \frac{1}{\binom{M}{N-1}} \left(S(B) + \sum_{k=1}^{N-1} \frac{\binom{N}{k}}{\binom{M}{k}} \right)$$

当且仅当 $S(B)$有正确的值时它可等于 $1\Big/\binom{M}{N}$。 ▌

外部查找　散列技术有助于对磁盘或磁鼓这样的直接存取存储设备进行外部查找。对于这样的应用,如同在 6.2.4 节中那样,我们要把对文件的存取次数极小化,这对于算法的选择有两项主要的影响:

1)花费较多的时间计算散列函数是合理的,因为坏的散列函数带来的损失,比起进行一项仔细的工作所需要的额外时间代价要大得多。

2)通常都把记录组织成页或桶,以便每次从外存取出若干个记录。

把文件分成为每个包含 b 个记录的 M 个桶。现在除非 b 个以上的键码有相同的散列地址,不然冲突不会造成问题。下列三种解决冲突的方法似乎是最好的:

A) *用分开的表列拉链*　如果有多于 b 个记录落到同一桶当中,则可在第一个桶的末尾插入一个通向"溢出"记录的链接。把这些溢出记录放在一个特殊的溢出区域中。通常,在溢出区域中存放桶并没有优点,因为溢出较少出现;于是,通常额外的记录链接在一起,使得一个表列的第$(b+k)$个记录需要 $1+k$ 次存取。通常一种好的想法是在一个磁盘文件的每个柱面上都为溢出保留某些为空间,使得大多数存取是对于相同柱面进行的。

尽管处理溢出的这个方法似乎低效,但溢出的次数从统计上说足够小,以致平均查找时间仍然非常好。见表 2 和表 3,它们示出了当 $M, N \to \infty$时,对于固定的 α,作为负载因子

$$\alpha = N/Mb \tag{54}$$

的函数的平均存取次数。奇怪的是,当 $\alpha = 1$ 时,不成功的查找的渐近存取次数竟随 b 而增加。

表 2 通过分开拉链进行一次不成功查找的平均存取数

桶	负载因子 α									
大小,b	10%	20%	30%	40%	50%	60%	70%	80%	90%	95%
1	1.0048	1.0187	1.0408	1.0703	1.1065	1.1488	1.197	1.249	1.307	1.34
2	1.0012	1.0088	1.0269	1.0581	1.1036	1.1638	1.238	1.327	1.428	1.48
3	1.0003	1.0038	1.0162	1.0433	1.0898	1.1588	1.252	1.369	1.509	1.59
4	1.0001	1.0016	1.0095	1.0314	1.0751	1.1476	1.253	1.394	1.571	1.67
5	1.0000	1.0007	1.0056	1.0225	1.0619	1.1346	1.249	1.410	1.620	1.74
10	1.0000	1.0000	1.0004	1.0041	1.0222	1.0773	1.201	1.426	1.773	2.00
20	1.0000	1.0000	1.0000	1.0001	1.0028	1.0234	1.113	1.367	1.898	2.29
50	1.0000	1.0000	1.0000	1.0000	1.0000	1.0007	1.018	1.182	1.920	2.70

表 3 通过分开拉链进行一次成功查找的平均存取数

桶	负载因子 α									
大小,b	10%	20%	30%	40%	50%	60%	70%	80%	90%	95%
1	1.0500	1.1000	1.1500	1.2000	1.2500	1.3000	1.350	1.400	1.450	1.48
2	1.0063	1.0242	1.0520	1.0883	1.1321	1.1823	1.238	1.299	1.364	1.40
3	1.0010	1.0071	1.0216	1.0458	1.0806	1.1259	1.181	1.246	1.319	1.36
4	1.0002	1.0023	1.0097	1.0257	1.0527	1.0922	1.145	1.211	1.290	1.33
5	1.0000	1.0008	1.0046	1.0151	1.0358	1.0699	1.119	1.186	1.286	1.32
10	1.0000	1.0000	1.0002	1.0015	1.0070	1.0226	1.056	1.115	1.206	1.27
20	1.0000	1.0000	10000	1.0000	1.0005	1.0038	1.018	1.059	1.150	1.22
50	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.001	1.015	1.083	1.16

B)**通过接合表列进行拉链** 我们可修改算法 C,使之适用于外部文件,以代替分开的溢出区域。对于每一个柱面,使用可利用空间的双重拉链表列,可把所有还未满的桶链接在一起。在这个方案中,每个桶包含一个计数,它表明桶中有多少记录位置为空,仅当计数变成 0 时,这个桶才从双重链接表中撤销。可用"活动指针"分布溢出(参见习题 2.5-6),使得不同的表列趋于使用不同的溢出桶。对这个方法还未曾分析过,但它将被证明是十分有用的。

C)**开式寻址法** 我们也可以不用链,而使用"开式"方法。当考虑外部查找时,线性探查大概比随机探查更好,因为通常可以选择增量 c 使它把连续存取间的等待延迟时间极小化。可推广上面作出的线性探查的近似理论模型,以考虑桶的影响,它表明,除非这个表已经变得非常满,否则线性探查确实令人满意。例如,参见表 4;当负载因子是 90% 且桶的大小是 50 时,在一次成功的查找中平均存取次数只是

1.04。这实际上比对于相同大小的桶使用拉链方法(A)所需要的 1.08 次存取更好！

表 4　通过线性探查在一次成功查找中的平均存取次数

桶大小,b	负载因子 α 10%	20%	30%	40%	50%	60%	70%	80%	90%	95%
1	1.0556	1.1250	1.2143	1.3333	1.5000	1.7500	2.167	3.000	5.500	10.50
2	1.0062	1.0242	1.0553	1.1033	1.1767	1.2930	1.494	1.903	3.147	5.64
3	1.0009	1.0066	1.0201	1.0450	1.0872	1.1584	1.286	1.554	2.378	4.04
4	1.0001	1.0021	1.0058	1.0227	1.0497	1.0984	1.190	1.386	2.000	3.24
5	1.0000	1.0007	1.0039	1.0124	1.0307	1.0661	1.136	1.289	1.777	2.77
10	1.0000	1.0000	1.0001	1.0011	1.0047	1.0154	1.042	1.110	1.345	1.84
20	1.0000	1.0000	1.0000	1.0000	1.0003	1.0020	1.010	1.036	1.144	1.39
50	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.001	1.005	1.040	1.13

对于方法(A)和(C)的分析涉及某些非常有趣的数学；我们在这里仅仅概述其结果，因为细节已在习题 49 和 55 中加以研究。这些公式涉及到同定理 K 的 Q 函数密切有关的两个函数，即

$$R(\alpha,n)=\frac{n}{n+1}+\frac{n^2\alpha}{(n+1)(n+2)}+\frac{n^3\alpha^2}{(n+1)(n+2)(n+3)}+\cdots \tag{55}$$

以及

$$t_n(\alpha)=\mathrm{e}^{-n\alpha}\left(\frac{(\alpha n)^n}{(n+1)!}+2\frac{(\alpha n)^{n+1}}{(n+2)!}+3\frac{(\alpha n)^{n+2}}{(n+3)!}+\cdots\right)=$$
$$\frac{\mathrm{e}^{-n\alpha}n^n\alpha^n}{n!}(1-(1-\alpha)R(\alpha,n)) \tag{56}$$

若用这些函数来表示，当 $M,N\to\infty$时，在一次不成功的查找中通过拉链方法(A)所作的平均存取次数是

$$C'_N=1+\alpha bt_b(\alpha)+O\left(\frac{1}{M}\right) \tag{57}$$

而在一次成功的查找中对应的次数是

$$C_N=1+\left(\frac{\mathrm{e}^{-b\alpha}b^b\alpha^b}{2b!}\right)(2+(\alpha-1)b+$$
$$(\alpha^2+(\alpha-1)^2(b-1))R(\alpha,b))+O\left(\frac{1}{M}\right) \tag{58}$$

这些公式的极限值是在表 2 和表 3 中所示的量。

由于拉链方法(A)需要一个分开的溢出区，因此我们需要估计将出现多少溢出。溢出的平均次数将是 $M(C'_N-1)=Nt_b(\alpha)$，因为 C'_N-1 是在任何给定的表列中平均的溢出次数。因此表 2 可用来推导出所需要的溢出空间的数量。对于固定的 α，当 $M\to\infty$时，溢出的总数的标准差将大致同$\sqrt{M}$成正比。

C'_N 和C_N 的渐近值出现在习题53中，但当 b 很小或α 很大时，这些近似不是非常好；幸而，$R(\alpha,n)$的级数收敛速度甚至当 α 很大时也颇快，所以这些公式可以没有多大困难地计算到任何想要的精度。由 Stirling 近似公式和 1.2.11.3 小节中对函数 $R(n)=R(1,n)-1$ 的分析，当 $b\to\infty$时，在

$$\max C'_N = 1+\frac{e^{-b}b^{b+1}}{b!}=\sqrt{\frac{b}{2\pi}}+1+O(b^{-1/2}) \tag{59}$$

$$\max C_N = 1+\frac{e^{-b}b^{b}}{2b!}(R(b)+1)=\frac{5}{4}+\sqrt{\frac{2}{9\pi b}}+O(b^{-1}) \tag{60}$$

的条件下，对于 $\alpha=1$ 出现极大值。

在通过线性探查所作的一次成功的外部查找中，平均的存取次数有异常简单的表达式

$$C_N\approx 1+t_b(\alpha)+t_{2b}(\alpha)+t_{3b}(\alpha)+\cdots \tag{61}$$

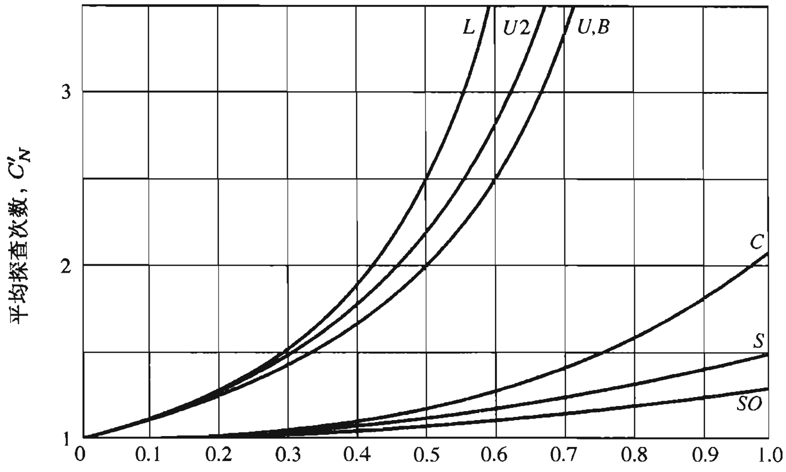
它可被理解如下：查找所有 N 个键码的平均总访问次数是NC_N，此即 $N+T_1+T_2+\cdots$，其中 T_k 是需要多于k 次访问的键码的平均个数。定理P指出，我们可以以任何顺序记入键码而不影响 C_N，并由此得出，如果我们有大小为 kb 的M/k 个桶，则 T_k 是在拉链方法中出现的溢出记录的平均个数，如果按我们上边所说的，也就是 $Nt_{kb}(\alpha)$。关于等式(61)的进一步论证请见习题55。

Charles A. Olson 已经精采地论述了早期在外部散列表的设计中所涉及的实用性的考虑，参见*Proc. ACM Nat. Conf.* **24**(1969)，539～549。他列举了若干实践过的例子，并且指出，如果文件受到经常的插入/删去活动的影响而不浮动记录，则溢出记录的次数将大量增加；他还给出了对于这一情况的分析。这是在 J. A. de Peyster 的参与下得到的。

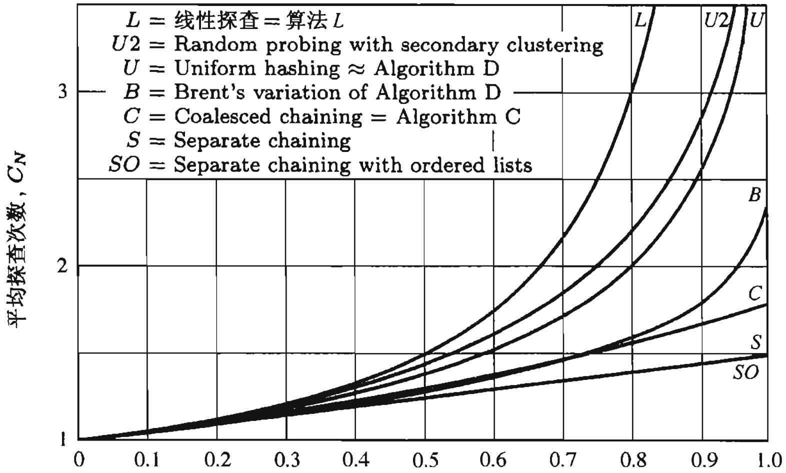
诸方法比较 我们现在已经研究了大量的用于查找的技术；怎样选出正适合于给定应用的一个方法呢？很难以几句话来概括在选择一个查找方法时所涉及的有关"折衷"的细节，但是相对于查找速度和所需的存储空间说来，下列诸事似乎是具有第一位重要性的。

图44概括了这一节的分析，并表明了解决冲突的各种方法导致的不同的探查次数。但是探查计数并未说出事情的全部，因为在不同的方法中每次探查的时间也不同，而且方法的变化对于运行时间有相当大的影响(如同我们在图42中见到的)。线性探查比图44中所示的其它方法更经常地访问表，但是它有简便的优点。而且线性探查也并非坏得可怕：当表有90%满时，为把一个随机项放置在表中，算法L平均需要少于5.5次的探查。(然而，当表90%满时，通过算法L插入一个新的项所需要的平均探查次数是50.5。)

图44表明，就探查次数而言，拉链方法是十分经济的，但因为链接字段需要额外的存储空间，故有时开式寻址法对于小记录更有吸引力。例如，如果我们需要在容量为500的一个拉链散列表和容量为1000的一个开式散列表之间进行选择，则后者显然是更可取的，因为当存有500个记录时，它的查找效率比较高，而且它有能



(a)



(b)

图 44 解决冲突诸方法的比较：当 $M\to\infty$ 时平均的探查次数的极限值

(a)不成功的查找；(b)成功的查找。

力吸收两倍的数据。另一方面，有些记录的大小和格式实际上无需额外的代价就能为链接字段提供空间(参见习题 65)。

怎样把散列方法同我们在本章中已研究过的其它查找策略作比较呢？从速度的观点看，我们可以论证，当记录个数很大时，散列法更好些，因为如果假设表不太满，则当 $N\to\infty$ 时，一个散列方法的平均查找时间保持有界。例如，当表有 90% 满

时,对于一次成功的查找,程序 L 将仅仅花费大约 55 个时间单位;当 N 大于 600 左右时,这比我们见到的最快的 MIX 二分查找程序要强(见习题 6.2.1-24),而代价仅为多花 11% 的存储空间。而且,二分查找仅适合于固定的表,而一个散列表却允许有效的插入。

我们也可以把程序 L 同允许动态插入的面向树的查找方法进行比较,当 N 大于约 90 时,对于 90% 满的表,程序 L 比程序 6.2.2T 更快;而当 N 大于约 75 时,它也比程序 6.3D 更快(习题 6.3-9)。

对于成功的查找说来,这一章中仅有一个查找方法是有效且实际上没有存储开销的,这就是算法 D 的 Brent 变形。他的方法允许我们把 N 个记录放到大小为 $M = N+1$ 的一个表中,而且平均以大约 2.5 次探查找出任何记录,链接字段不需要额外的空间等;然而,对于不成功的查找将是非常缓慢的,需要大约 $N/2$ 次的探查。

因此,散列有若干优点。另一方面,在三个重要的方面散列表的查找比我们已经讨论的其它方法要差:

a)在散列表中一次不成功的查找之后,我们仅仅知道所希望的键码不存在。以比较为基础的查找方法总是产生更多的信息,它们允许我们找出 $\leqslant K$ 的最大键码和/或 $\geqslant K$ 的最小键码;这在许多应用中很重要的;例如,它允许我们对一个存储好的函数值表进行内插。也可使用以比较为基础的算法,来找出处于两个给定的值 K 和 K' 之间的所有键码。而且 6.2 节的树查找方法使得以递增的次序来遍历一个表的内容很容易做,而无需对它单独排序。

b)对散列表进行存储分配通常有些困难的;我们需要提供某一个存储区域用来存散列表,但应当分配多少空间可能不明显。如果我们提供了太多的内存,则可能损害其它表列或其它计算机用户范围内的存储;但如果不提供足够的空间,则这个表会溢出。反之,树查找和插入算法所处理的树不会增长得比需要的更大。在一个虚拟的存储环境中,如果我们使用树查找或数字树查找,就可使内存访问局部化,而不是去建立一个很大的散列表,散列表几乎在每次散列一个键码时,就要求操作系统访问一个新的页。

c)最后,当我们使用散列方法时,需要对概率论有巨大的信赖,因为散列法仅仅平均来说是有效的,而最坏的情况则是可怕的!与随机数生成程序的情况类似,我们决不能完全保证当把一个散列函数应用于一个新的数据集合时,它能工作得很好。因此散列存储对于某些同人类的生命攸关的实时应用,如空间交通控制,将是不适当的。6.2.3 小节和 6.2.4 小节的平衡树算法更为安全,因为它们的查找时间的上限是有保证的。

历史 散列的思想看来是由 H. P. Luhn 创立的,他于 1953 年 1 月写了一篇 IBM 的内部备忘录,建议使用拉链;事实上,这也是拉链线性表的最初应用之一。他提出了在外部查找中以使用包含一个以上元素的桶为好。不久以后,A. D. Lin 把 Luhn 的分析推进一步,并且提出了使用“退化地址”处理溢出的技术;例如,假定存在 10000 个初级桶,1000 个二级桶,100 个三级桶,等等,则由初级桶 2748 产生的溢

出被置于二级桶 274 中；而从该桶产生的溢出再转到三级桶 27 中，等等。Luhn 最初提出的散列函数本质上是数字的；例如，他把键码数字的相邻对加起来并 mod 10，使得 31415926 被压缩成 4548。

大约同一时间，散列的思想独立地出现于 IBM 的另一伙成员当中。他们是：Gene M. Amdahl、Elaine M. Boehme、N. Rochester 以及 Arthur L. Samuel，他们编制了对于 IBM 701 的一个汇编程序。为了处理冲突问题，Amdanl 创立了使用线性探查的开式寻址法的思想。

散列代码首先是由 Arnold. I. Dumey 第一个在公开的文献中描述的，参见 *Computers and Automation* **5**，12(1956 年 12 月)，6～9。他第一个提出除以一个素数和使用余数作为散列地址的思想。Dumey 有趣的论文提到了拉链而非开式寻址法。俄罗斯的 P. Ershov 于 1957 年独立地发现了线性开式寻址法[*Doklady Akad. Nauk SSSR* **118** (1958)，427～430]；他发表了关于探查次数的经验结果，正确地猜测到当 $N/M<2/3$ 时，每一成功的查找的平均探查次数<2。

W. W. Peterson 的一篇经典的文章，*IBM J. Research & Development* **1** (1957)，130～146，乃是第一篇讨论在大型文件中的查找问题的重要文章。Peterson 一般地定义了开式寻址法，分析了均匀散列的性能，而且对于各种桶的大小给出了关于线性开式寻址法的各种经验统计，说明当删去项目时出现的性能上的蜕化。六年之后，Werner Buchholz[*IBM Systems J.* **2** (1963)，86～111] 发表了关于这个课题的另一篇广泛的综述，他给出了关于散列函数的一个特别好的讨论。A. G. Konheim 和 B. Weiss，*SIAM J. Appl. Math.* **14** (1966)，1266 ～ 1274；V. Podderjugin，*Wissenschaftliche Zeitschrift der Technischen Universität Dresden* **17** (1968)，1087～1089 首先发表了对于算法 L 的正确的分析。

到这时为止，线性探查是出现于文献中的惟一的一类开式寻址法方案，但还有一个通过独立的散列函数进行重复的随机探查为基础的方案是由另外一些人独立地提出来的(见习题 48)。在之后数年中散列广泛地被使用，但是几乎没有发表任何更多的东西。然后 Robert Morris 写了一篇关于这个课题的非常有影响的综述[*CACM* **11** (1968)，38～44]，其中他介绍了带有二次堆积的随机探查的思想。Morris 的文章引起了一阵研究热潮，并以算法 D 及其改进而达到高潮。

说明这样一点是有趣的，具有目前的意义的“散列”(hashing)一词，在 20 世纪 60 年代末期以前，似乎完全没有见诸文字当中，尽管那时在世界的若干地区它已经变成了普遍的术语。这个字最初见于书面似乎是在 H. Hellerman 的书*Digital Computer System Principles* (纽约：McGraw－Hill，1967)，第 152 页中。在撰写这一节时，我所研究的近 60 篇有关的文献当中，这一词惟一地出现于 1961 年由 W. W. Peterson 所写的一篇未发表的备忘录中。在 20 世纪 60 年代中期，不知怎么回事，“散列”(to hash)这一动词竟魔术般地变成了关于键码转换的标准术语，但是在 1967 年以前却没有人敢很轻率地公开使用这样一个不庄重的词。

最新的进展 自从作者于 1972 年最初编写这一章时，散列在理论和时间上都

已取得许多进展，但上面讨论的基本思想对于通常的应用来说仍然是有用的。例如，由 J. S. Vitter 和陈文进所写的书*Design and Analysis of Coalesced Hashing*（纽约：牛津大学出版社，1987）讨论和分析了算法 C 的若干有教益的变形。

从一种实用的观点看，在 20 世纪 70 年代末期发明的最重要的散列技术大概是 Witold Litwin 称之为线性散列的方法［*Proc. 6th International Conf. on Very Large Databases*（1980），212～223］。线性散列——顺便说它和线性探查的经典技术毫无关系——当诸项被插入和/或删去时，允许散列地址的数目增长和/或合适地收缩。Per-Ake Larson 在*CACM* **31**（1988），446～457 给出了关于线性散列，（包括对于内查找，它同其它方法的比较在内）的一个杰出的讨论；对于当许多大的和/或小的表同时出现时的改进，也请见 W. G. Griswold 和 G. M. Townsend，*Software Practice & Exp.* **23**（1993），351～367。线性散列也被用于在一个网络的许多不同的现场之间分布的巨大的数据库中［参见 Litwin，Neimat 和 Schneider，*ACM Trans. Database Syst.* **21**（1996），480～525］。大约在同一时期，R. Fagin，J. Nievergelt，N. Pippenger 和 H. R. Strong［*ACM Trans. Database Syst.* **4**（1979），315～344］提出了称作**可扩充的散列**的另一个方案。它有这样的性质，即为检索任何记录，至多需要两次访问外部的页。当键码的次序不重要时，线性散列和可扩充散列都比 6.2.4 小节的 B 树更可取。

在理论领域中，已经设计出更复杂的方法，通过这些方法，有可能保证，不论被考察的键码如何，对于每次访问，有 $O(1)$的极大时间，且对于每个插入和删除有 $O(1)$的平均平摊时间；而且，在任何时间里所使用的总共的存储以一个常数乘上当前存在的次数为限，加上另外一个加性常数。在 Fredman，Komlós 和 Szemerédi［*JACM* **31**（1984），538～544］的思想为基础构建出来的这一结果，是由 Dietzfelbinger, Karlin, Mehlhorn, Meyer auf der Heide, Rohnert 以及 Tarjan［*SICOMP* **23**（1994），738～761］给出的。

习 题

1.［*20*］假定 K 的字节 1，2，3 各包含少于 30 个字母字符代码，当达到表 1 中的指令 9H 时，rI1 的内容可以多小和多大？

2.［*20*］找出可以附加到表 1 而又无需改变程序的相当常用的英文字。

3.［*23*］说明为什么对任何常数 a，以下列五条指令开始的程序

```
LD1 K(1:1)   或   LD1N K(1:1)
LD2 K(2:2)   或   LD2N K(2:2)
INC1 a,2
LD2 K(3:3)
J2Z 9F
```

都不能用来代替表 1 中更复杂的程序，因为对于给定的键码将不能产生惟一的地址。

4.［*M30*］为了使得很可能有三个人有相同的生日，应该有多少人参加一个晚会？

5.［*15*］B. C. Dull 曾经用一台十进的 MIX 计算机写一个 FORTRAN 编译程序，而且他需要一

个符号表来记住正在被编译的 FORTRAN 程序中的变量名字。这些名字的长度限制成至多 10 个字符。他决定使用 $M=100$ 的一个散列表，而且使用快速散列函数 $h(K)=K$ 的最左字节。这是一个好的想法吗？

6.[*15*] 把(3)的头两条指令改变成为 LDA K;ENTX 0，是明智的吗？

7.[*HM30*]（*多项式散列*）本题的目的是考虑如像(10)那样的多项式 $P(x)$ 的构造，它把 n 位键码转换成为 m 位地址，使得任何两个不相同的键码如果其区别不超过七位，则它们一定被散列成不同的地址。给定 n 和 $t\leqslant n$，并给定一个整数 k，使得 n 整除 α^k-1，我们将构造一个其次数 m 是 n，t 和 k 的函数的一个多项式（通常在必要时增大 n 使得把 k 选择成相当小）。

设 S 是使得 $\{1,2,\cdots,t\}\subseteq S$ 的整数的最小集合，而且设对所有 $j\in S$，$(2j)\bmod n\in S$。例如，当 $n=15,k=4,t=6$ 时，我们有 $S=\{1,2,3,4,5,6,8,10,12,9\}$。现在定义多项式 $P(x)=\Pi_{j\in S}(x-\alpha^j)$，其中 α 是有限域 $GF(2^k)$ 中阶为 n 的一个元素，而且 $P(x)$ 的系数是在这个域中计算的。$P(x)$ 的次数 m 是 S 的元素的个数。因为当 α^j 是 $P(x)$ 的一个根时，α^{2j} 也是它的一个根，由此得出 $P(x)$ 的系数 p_i 满足 $p_i^2=p_i$，所以它们全都为 0 或 1。

试证明，如果 $R(x)=r_{n-1}x^{n-1}+\cdots+r_1x+r_0$ 是任意非 0 多项式 modulo 2，而且至多有 t 个非 0 的系数，则 $R(x)$ 不是 $P(x)$ modulo 2 的一个倍数［由此得出，对应的散列函数的特性就像已宣告的那样。］

8.[*M34*]（*三距离定理*）设 θ 是 0 和 1 之间的无理数，在 4.5.3 小节的记号下它的连分式表示是 $\theta=/\!/a_1,a_2,a_3,\cdots/\!/$。设 $q_0=0,p_0=1,q_1=1,p_1=0$，对于 $k\geqslant1$，$q_{k+1}=a_kq_k+q_{k-1}$，$p_{k+1}=a_kp_k+p_{k-1}$。设 $\{x\}$ 表示 $x\bmod 1=x-\lfloor x\rfloor$，且设 $\{x\}^+$ 表示 $x-\lceil x\rceil+1$。随着点 $\{\theta\}$，$\{2\theta\}$，$\{3\theta\}$，…逐次插入到区间$[0,1]$中，而把线段按它们出现的方式进行编号，使得一个给定长度的第一段编号为 0，第二段为 1，等等。试证明下列命题全为真：长度为 $\{t\theta\}$ 且编号为 s 的区间，其左端点为 $\{s\theta\}$，右端点为 $\{(s+t)\theta\}^+$，其中 $t=rq_k+q_{k-1}$ 和 $0\leqslant r<a_k$ 且 k 为偶数以及 $0\leqslant s<q_k$。长度为 $1-\{t\theta\}$ 且编号为 s 的区间，其左端点为 $\{(s+t)\theta\}$，右端点为 $\{s\theta\}^+$，其中 $t=rq_k+q_{k-1}$，和 $0\leqslant r<a_k$ 且 k 为奇数及 $0\leqslant s<q_k$。每个正整数 n 均可惟一地表示成 $n=rq_k+q_{k-1}+s$，其中 $k\geqslant1$，$1\leqslant r\leqslant a_k$，$0\leqslant s<q_k$。借助这个表示，在点 $\{n\theta\}$ 被插入之前，已有的 n 个区间为

长度 $\{(-1)^k(rq_k+q_{k-1})\theta\}$ 的头 s 个区间（编号 $0,\cdots,s-1$）；

长度 $\{(-1)^{k+1}q_k\theta\}$ 的头 $n-q_k$ 个区间（编号为 $0,\cdots,n-q_k-1$）；

长度 $\{(-1)^k((r-1)q_k+q_{k-1})\theta\}^+$ 的最后 q_k-s 个区间（编号为 $s,\cdots,q_k-1$）

插入 $\{n\theta\}$ 的操作撤销编号为 s 的最后一种类型的区间，并把它转换成编号为 s 的第一种类型的区间，以及编号为 $n-q_k$ 的第二种类型的区间。

9.[*M30*] 当我们逐次地把点 $\{\theta\}$，$\{2\theta\}$，…插入到区间$[0..1]$中时，定理 S 断言每个新的点总是分割剩下的最大的区间之一。如果把区间$[a..c]$分成两个部分$[a..b]$，$[b..c]$，而其中的一个部分大于另一部分的两倍长，即若 $b-a>2(c-b)$ 或 $c-b>2(b-a)$，则我们称它为一个坏的分割。

证明除非 $\theta\bmod 1=\phi^{-1}$ 或 ϕ^{-2}，对于某个 $\{n\theta\}$ 将出现坏的分割，而当 $\theta\bmod 1=\phi^{-1}$ 或 ϕ^{-2} 时，将决不产生坏的分割。

10.[*M38*]（R.L.Graham）如果 $\theta,\alpha_1,\cdots,\alpha_d$ 是实数且 $\alpha_1=0$，且如果 $n_1,\cdots,n_d$ 是正整数，此外如果对于 $0\leqslant n<n_j$，$1\leqslant j\leqslant d$，点 $\{n\theta+\alpha_j\}$ 被插入到区间$[0..1]$中，则得到的 $n_1+\cdots+n_d$（可能为空）个区间至多有 $3d$ 个不同的长度。

11.[*16*] 成功的查找通常比不成功的查找要更为频繁。如果因此而把程序 C 的行 12～13 同

行 10～11 加以交换，这是否为一个好想法？

▶12. [*21*] 证明程序 C 可被重写成使得在内部循环中仅有一个条件转移。把修改了的程序同原来的程序的运行时间作比较。

▶13. [*24*] (**略写的键码**) 设 $h(K)$是一个散列函数，且设 $q(K)$是 K 的一个函数，使得一旦已经给定 $h(K)$和 $q(K)$，便可确定 K。例如，在除法散列中，我们可以设 $h(K)=K \bmod M$ 及 $q(K)=\lfloor K/M\rfloor$；在乘法散列中可以设 $h(K)$是$(AK/w) \bmod 1$ 的前几个二进位，而 $q(K)$是其它的二进位。

试证明，当使用拉链而不使用重叠的表列时，在每个记录中我们仅需存储 $q(K)$而不是 K(这几乎节省了链接字段所需要的空间)。试修改算法 C 使其避免重叠的表列，以允许这样略写的键码，而且对于“溢出”的记录还不使用辅助存储单元。

14. [*24*] E.W.Elcock 证明有可能让一个很大的散列表同任意多个其它拉链表共享内存。设这个表列区域的每个字有一个 2 位二进位的 TAG 字段以及称作 LINK 和 AUX 的两个链接字段，并有下列的解释：

TAG(P) = 0 表示可用空间表列中的一个字；LINK(P)指向该表中的下一项，AUX(P)不用。

TAG(P) = 1 表示正在使用的一个字，其中 P 不是散列表中任何键码的散列地址；单元 P 中字的其它字段可以有任何需要的格式。

TAG(P) = 2 表示 P 至少是一个键码的散列地址；AUX(P)指向确定所有这样键码的一个链接表，LINK(P)指向这个表存储器中的另一个字。在处理任何表期间，每当访问带有 TAG(P) = 2 的一个字时，就重复地置 P←LINK(P)，直到达到具有 TAG(P)≤1 的一个字为止。(为了效率，我们也可以改变以前的链，使得今后不需要一再地跳过这些散列表的项。)

为在这样的一个组合表中插入和检索键码，试定义一个适当的算法。

15. [*16*] 为什么算法 L 和算法 D 当 $N=M-1$ 而不是当 $N=M$ 时警告溢出是一个好想法？

16. [*10*] 程序 L 指出，K 不应为 0。但当 K 为 0 时，它是否就真正不能工作了？

17. [*15*] 当 $h_1(K)\neq 0$ 时，为什么不简单地在(25)中定义 $h_2(K)=h_1(K)$？

▶18. [*21*] 作为程序 D 的行 10～13 的一个替换，(31)比(30)好些还是坏些？试根据 A，$S1$ 和 C 的平均值，给出你的答案。

19. [*40*] 试凭经验测试算法 D 中以下列方式限制 $h_2(K)$的范围的效果：(a)对于 $r=1,2,3,\cdots 10$，使得 $1\leq h_2(K)\leq r$；(b)对于 $\rho=\frac{1}{10},\frac{2}{10},\cdots,\frac{9}{10}$，使得 $1\leq h_2(K)\leq \rho M$。

20. [*M25*] (R.Krutar) 改变算法 D 如下以免去散列函数 $h_2(K)$：在步骤 D3 中置 $c\leftarrow 0$；且在步骤 D4 的开始置 $c\leftarrow c+1$。试证明，如果 $M=2^m$，则对应的探查序列 $h_1(K)$，$(h_1(K)-1) \bmod M,\cdots,\left(h_1(K)-\binom{M}{2}\right) \bmod M$ 将是$\{0,1,\cdots,M-1\}$的一个排列。假定这个方法的特性与具有二次堆积的随机探查一样，则当把这个二次探查方法编成 MIX 程序时，该程序与图 42 中考虑的三个程序比起来是好还是坏？

▶21. [*20*] 假设我们希望从由算法 D 构造的一个表中删去一个记录，并如同正文中所建议的那样，标记它为“删去的”。试问，我们也应该把用以支配算法 D 的变量 N 减值吗？

22. [*27*] 证明算法 R 使表保持好像它未曾在开头的位置被插入过 KEY[i]那样。

▶23. [*33*] 试设计类似于算法 R 的一个算法，用于从通过算法 C 构造的一个拉链散列表中删去项。

24. [*M20*] 假设所有可能出现的键码的集合有 MP 个元素，其中恰有 P 个键码散列到任何给定的地址。(在实际情况下，P 是非常大的；例如当诸键码为任意的十进数字时，如果 $M=10^3$，

则我们有 $P=10^7$。)假设 $M\geqslant 7$ 且 $N=7$。如果从所有可能的键码的集合中随机地选择七个不同的键码,则作为 M 和 P 的函数得到的散列序列为 1 2 6 2 1 6 1(即 $h(K_1)=1, h(K_2)=2, \cdots, h(K_7)=1$)的精确概率等于多少?

25.[*M19*]　说明等式(39)为什么是正确的?

26.[*M20*]　利用线性探查时,有多少散列序列产生如(21)那样的已占用单元的型式。

27.[*M27*]　完成定理 K 的证明[提示:设

$$s(n,x,y)=\sum_k \binom{n}{k}(x+k)^{k+1}(y-k)^{n-k-1}(y-n)$$

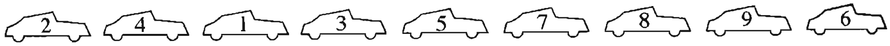
使用 Abel 二项式定理,即等式 1.2.6-(16),来证明 $s(n,x,y)=x(x+y)^n+ns(n-1,x+1,y-1)$]。

28.[*M30*]　昔日的计算机比今天的慢得多,有可能观看指示灯的闪烁,并了解算法 L 运行多快。当表开始填满时,某些项将非常快地被处理,而其它的则花费大量的时间。

这个经验提示,当使用线性探查时在一次不成功的查找中探查次数的标准差是相当高的。试求一个公式,它借助于在定理 K 中定义的 Q_r 函数表达方差,并估计当 $M\rightarrow\infty$ 且 $N=\alpha M$ 时的方差。

29.[*M21*]　(停车问题)某单行道有排成一行的 m 个停放车辆的位置,其编号为 1 到 m。一个人和他的打瞌睡的妻子驱车过街,突然他妻子醒过来并命令他立即停车,他顺从地把车停在第一个可利用的位置;但如果没有剩下位置了,则他可以不必后退(即如果在他车子达到位置 k 时,他的妻子才醒过来,但位置 $k, k+1, \cdots, m$ 全都满了),他表示歉意并继续行驶。

事实上,假设对 n 辆不同的小汽车发生此事,其中第 j 个妻子刚巧在停车位置 a_j 时醒过来。假定这条街道开始时是空的,而且在停车之后没有人离开,试问有多少个序列 $a_1\cdots a_n$ 使所有的小汽车都能安全地停放?例如,当 $m=n=9$ 和 $a_1\cdots a_9=3\ 1\ 4\ 1\ 5\ 9\ 2\ 6\ 5$ 时,这些小汽车停放如下:



[提示:使用对线性探查的分析。]

30.[*M28*]　在习题 29 的停车问题中,当 $n=m$ 时,试证明所有的小汽车都能停放的充要条件是存在 $\{1,2,\cdots,n\}$ 的一个排列 $p_1p_2\cdots p_n$,使得对所有的 j, $a_j\leqslant p_j$。

31.[*M40*]　在习题 29 的停车问题中,当 $n=m$ 时,解的个数已证明是 $(n+1)^{n-1}$;而且从习题 2.3.4.4-22 我们知道,这和具有 $(n+1)$ 个带标号顶点的自由树的个数相同!试求在停车序列和树之间的有趣联系。

32.[*M26*]　证明每当 $b_0, b_1, \cdots, b_{M-1}$ 是和数小于 M 的非负整数时,方程组(44)有惟一解 $(c_0, c_1, \cdots, c_{M-1})$。试设计出求这个解的算法。

▶33.[*M23*]　说明(51)为什么仅仅是由算法 L 所作的真正的平均探查次数的一个近似。在(5′)的推导中有什么不是严格地精确的东西?

▶34.[*M22*]　本题的目的是研究:当诸表列像(38)那样保持分开时,在一个拉链的散列表中的平均探查次数。

a)求 P_{Nk},即当 M^N 个散列序列(35)同等可能时,一个给定的表列有长度 k 的概率。

b)求生成函数 $P_n(z)=\sum_{k\geqslant 0}P_{Nk}z^k$。

c)借助于这个生成函数,表达出一次成功的查找所需的平均探查次数。

d)推导出在一次不成功的查找中的平均探查次数,并考虑采用下列约定的数据结构变形:(i)

散列总是对一个表头进行(参见图 38);(ii)散列是对表的一个位置进行(参见图 40),但除了一个表的开头之外所有的键码进入到一个分开的溢出区域;(iii)散列是对表的一个位置进行而且所有项都出现于散列表中。

35.[*M24*] 继续习题 34,当各表列按照它们的键码值保持有序时,在一次不成功的查找当中平均探查次数是多少?考虑数据结构(i)、(ii)和(iii)。

36.[*M23*] 继续习题 34(d),当查找不成功时,使用数据结构(i)和(ii),求探查次数的*方差*。

▶37.[*M29*] 公式(19)给出当查找成功时在分开的拉链中探查的平均次数;这个量的*方差*是多少?

38.[*M32*](*树散列*) 一个机灵的程序员可以尝试使用二分查找树以代替拉链方法中的线性列表,由此把算法 6.2.2T 与散列结合起来。试分析通过这个复合的算法,对于成功和不成功的查找两者所需要的平均探查次数。[*提示*:参看式 5.2.1-(15)]。

39.[*M28*] 设 $c_N(k)$是当把算法 C 应用于所有 M^N 个散列序列(35)时形成的长度为 k 的列表总数。求关于数 $c_N(k)$的一个递推关系使得有可能确定对于和

$$S_N = \sum_k \binom{k}{2} c_N(k)$$

的一个简单公式。S_N 与用算法 C 作不成功查找时所需平均探查数的关系如何?

40.[*M33*] 公式(15)给出在一次不成功的查找中由算法 C 所使用的平均探查次数,这个量的*方差*是多少?

41.[*M40*] 分析 T_N,即当通过算法 C 插入第 $N+1$ 个项目时下标 R 的值减 1 的平均次数。

▶42.[*M20*] 推导(17),即算法 C 立即成功的概率。

43.[*HM44*] 试分析使用大小为 $M'\geqslant M$ 的一个表对算法 C 进行的一项修正。在这项修正中仅仅前 M 个单元被用于散列,所以在步骤 C5 中找到的前 $M'-M$ 个空的节点将在这个表的额外单元中。对于固定的 M',在 $1\leqslant M\leqslant M'$范围中选择什么样的 M 方能导致最好的性能?

44.[*M43*](*有二次堆积的随机探查*) 这个习题的目标是确定具有探查序列

$$h(K),\ (h(K)+p_1) \bmod M,\ (h(K)+p_2) \bmod M, \cdots,$$
$$(h(K)+p_{M-1}) \bmod M$$

的开式寻址方案中预期的探查次数,其中 $p_1p_2\cdots p_{M-1}$是依赖于 $h(K)$的一个随机选择的$\{1,2,\cdots,M-1\}$的排列。换言之,具有相同 $h(K)$值的所有键码都遵循相同的探查序列,而且具有这一性质的 M 个探查序列的$(M-1)!^M$ 种可能的选择都是同等可能的。

这一情况可以通过对开始时是空的大小为 m 的线性阵列实施的下列实验步骤来精确地模拟。进行如下操作 n 次:“以概率 p,占用最左的空位置,否则(即以概率 $q=1-p$),选择表中除最左者外的任意位置,而且这 $m-1$ 个供选择的位置是同等可能的。如果选出的位置是空的,则占用它;否则选择任何空的位置(包括最左边的)并占用它,同时认为每一个空的位置是同等可能的。”

例如,当 $m=5$ 和 $n=3$ 时,在上述的实验之后的阵列配置将为(已占用,已占用,空,已占用,空)的概率是:

$$\frac{7}{192}qqq + \frac{1}{6}pqq + \frac{1}{6}qpq + \frac{11}{64}qqp + \frac{1}{3}ppq + \frac{1}{4}pqp + \frac{1}{4}qpp$$

(这个过程对应于当 $p=1/m$ 时的有二次堆积的随机探查,因为我们可以把表的项重新编号使得一个特殊的探查序列是 $0,1,2,\cdots$而所有其它的都是随机的。)

试求出在这个阵列左边的已占用位置(即,在上例中的 2)的平均数的一个公式。并求当 $p=1/m$, $n=\alpha(m+1)$和 $m\to\infty$时,这些量的渐近值。

45.[*M43*] 当探查序列以 $h_1(K)$,$(h_1(K)+h_2(K)) \bmod M$ 开始,且随后的探查仅仅依赖于 $h_1(K)$和 $h_2(K)$的随机选择时,求解具有三次堆积的习题 44 的类似情况。(于是,具有这一性质的 $M(M-1)$个探查序列的$(M-2)!^{M(M-1)}$种可能的选择,被认为是同等可能的。)这个过程渐近地等价于均匀探查吗?

46.[*M42*] 试确定使用探查序列

$$h(K),0,1,\cdots,h(K)-1,h(K)+1,\cdots,M-1$$

的开式寻址方法的 C'_N和 C_N。

47.[*M25*] 当探查序列是

$$h(K),h(K)-1,h(K)+1,h(K)-2,h(K)+2,\cdots$$

时,试求为开式寻址所需要的平均探查数。这个探查序列曾一度被提出,因为当 M 为偶数时,连续探查之间的所有距离都是不同的。[提示:找窍门,这个问题很容易。]

▶48.[*M21*] 给定互相独立的随机散列函数 $h_n(K)$的一个无穷序列,试分析探查位置 $h_1(K),h_2(K),h_3(K),\cdots$的开式寻址方法。在这个方案下,有可能探查同一个位置两次,例如如果 $h_1(K)=h_2(K)$,但在诸表变满之前,这样的偶然性很少可能。

49.[*HM24*] 推广习题 34 到每个桶有 b 个记录的情况,对于具有分开的表列的拉链,确定其平均的探查次数(即外部存储访问)C_N 和 C'_N,并假定在一次不成功的查找中含有 k 个元素的一个表列需要 $\max(1,k-b+1)$次探查。不像在习题 34 中那样使用精确的概率 P_{Nk},我们使用泊松近似

$$\binom{N}{k}\left(\frac{1}{M}\right)^k\left(1-\frac{1}{M}\right)^{N-k}=\frac{N}{M}\frac{N-1}{M}\cdots\frac{N-k+1}{M}\left(1-\frac{1}{M}\right)^N\left(1-\frac{1}{M}\right)^{-k}\frac{1}{k!}=$$
$$\frac{e^{-\rho}\rho^k}{k!}(1+O(k^2/M))$$

当 $M\to\infty$时,它对于 $N=\rho M$ 和 $k\leqslant\sqrt{M}$成立。试推导公式(57)和(58)。

50.[*M20*] 证明,在(42)的记号下,$Q_1(M,N)=M-(M-N-1)Q_0(M,N)$。[提示:首先证明 $Q_1(M,N)=(N+1)Q_0(M,N)-NQ_0(M,N-1)$]

51.[*HM17*] 借助于在(42)中定义的函数 Q_0,来表达在(55)中定义的函数 $R(\alpha,n)$。

52.[*HM20*] 证明 $Q_0(M,N)=\int_0^\infty e^{-t}(1+t/M)^N dt$。

53.[*HM20*] 证明函数 $R(\alpha,n)$可借助于不完备的伽马函数表达,并使用习题 1.2.11.3-9 的结果求出当 $n\to\infty$时,对于固定的 $\alpha<1$,$R(\alpha,n)$的渐近值直到 $O(n^{-2})$。

54.[*HM28*] 证明当 $b=1$ 时,等式(61)等价于等式(23)。提示:我们有

$$t_n(\alpha)=\frac{(-1)^{n-1}}{n!\alpha}\sum_{m>n}\frac{(-n\alpha)^m}{m(m-1)(m-n-1)!}$$

55.[*HM43*] 推广在定理 P 之后讨论的 Schay-Spruth 模型到大小为 b 的M 个桶的情况。证明 $C(z)$等于 $Q(z)/(B(z)-z^b)$,其中 $Q(z)$是次数为 b 的多项式且$Q(1)=0$。证明探查的平均次数是

$$1+\frac{M}{N}C'(1)=1+\frac{1}{b}\left(\frac{1}{1-q_1}+\cdots+\frac{1}{1-q_{b-1}}-\frac{1}{2}\frac{B''(1)-b(b-1)}{B'(1)-b}\right)$$

其中 $q_1,\cdots,q_{b-1}$是 $Q(z)/(z-1)$的根。以泊松近似式 $P(z)=e^{b\alpha(z-1)}$代替二项式概率分布 $B(z)$,其中 $\alpha=N/Mb$,使用 Lagrange 求逆公式(参见等式 2.3.4.4-(21)和习题 4.7-8),简化你的

答案成为等式(61)。

56.[*HM43*] 推广定理K,得到对于使用大小为 b 的桶的线性探查之精确分析。当这个表变满($N=Mb$)时,在一次成功的查找中探查的渐近次数是多少?

57.[*M47*] 若均匀地选定各探查序列的概率,是否能使 C_N 的值相对于所有的开式寻址方法来说成为极小?

58.[*M21*](S.C.Johnson) 求出在定理U的意义下等价于均匀散列的{0,1,2,3,4}的10个排列。

59.[*M25*] 证明,如果选定排列的一个概率,它在定理U的意义下等价于均匀散列,则当 M 充分大时,对于任何固定的指数 a,具有非0概率的排列数超过 M^a。

60.[*M47*] 如果一个开式寻址方案恰好使用 M 个探查序列,其中每个序列以 $h(K)$ 的每个可能的值开始,而每一个可能的值都以 $1/M$ 的概率出现,则称此方案为单散列方案。

问(在 C_N 为极小的意义下)最好的单散列方案是否渐近地比由(29)描述的随机方案更好?特别是当 $M\to\infty$ 时,是否 $C_{\alpha M}\geqslant 1+\frac{1}{2}\alpha+\frac{1}{2}\alpha^2+O(\alpha^3)$?

61.[*M46*] 在习题46中分析的方法,是否是在习题60意义下最坏的单散列方案?

62.[*M49*] 如果在习题44的记号下,对于所有的 K 增量 $p_1,p_2,\cdots,p_{M-1}$ 都是固定的,则称单散列方案是*循环的*。(这样的方法的例子是在习题20和47中考虑的线性探查和序列。)一个*最优的*单散列方案是这样的一个方案,对于一个给定的 M,对于所有$(M-1)!^M$ 个单散列方案说来,它的 C_M 是极小的。当 $M\leqslant 5$ 时,最好的单散列方案是循环的。这对于所有的 M 都成立吗?

63.[*M25*] 如果在散列表中反复地进行随机的插入和删去,则平均需要作多少次独立的插入,才能使所有 M 个位置至少被占用过一次?(这是对单元简单地标记为"已删去"的删除方法出现故障的平均时间)

64.[*M41*] 试分析算法R(通过线性探查的删去)的预期的特性。平均将实施步骤R4多少次?

▶65.[*20*] (可变长的键码) 在散列表的许多应用中,处理的是任意长字符串的键码。在这样的情况下,我们不能像在本节的程序中那样,把键码简单地存入表中。在MIX计算机上,在散列表中处理可变长键码的好方法应是怎样的?

▶66.[*25*](Ole Amble,1973) 在开式散列表中插入键码时能否利用它们的数值或字母顺序,使得用算法L或算法D进行查找时,只要一遇到小于查找变元的键码,便知查找已失败?

67.[*M41*] 如果算法L以分别的散列地址 $a_1a_2\cdots a_N$ 插入 N 个键码,令 d_j 是第 j 个键码同它的原地址 a_j 的位移量;则 $C_N=1+(d_1+d_2+\cdots+d_N)/N$。定理P告诉我们诸 a 的排列对于和 $d_1+d_2+\cdots d_N$ 没有影响。然而这样的排列却可能激烈地改变和 $d_1^2+d_2^2+\cdots+d_N^2$。例如,散列序列 $1\,2\cdots N-1\,N-1$ 使 $d_1d_2\cdots d_{N-1}d_N=0\;0\cdots 0\;N-1$ 和 $\sum d_j^2=(N-1)^2$,而它的反射 $N-1\;N-1\cdots 2\;1$ 导致温和得多的位移 $0\;1\cdots 1\;1$,对于它 $\sum d_j^2=N-1$。

a) $a_1a_2\cdots a_N$ 的哪一个重新安排使 $\sum d_j^2$ 极小化?

b)说明怎样来修改算法L使得在每次插入之后它维持位移量的一个极小方差集合?

c)确定带有和不带有这个修改的 $\sum d_j^2$ 的平均值。

68.[*M41*] 通过算法L的一次成功查找中平均探查次数的方差是多少?特别是,在习题67的记号下 $(d_1+d_2+\cdots+d_N)^2$ 的平均值是多少?

69.[*M25*](姚期智) 证明在习题62意义下的所有循环的单散列方案满足不等式 $C'_{\alpha M}\geqslant\frac{1}{2}(1+1/(1-\alpha))$。[提示:证明一个不成功的查找恰好花费 k 次探查的概率是 $p_k\leqslant(M-N)/$

M。]

70.[*HM43*]　试证明使用双散列为插入第($\alpha M+1$)项所需的预期探查数至多是使用均匀探查插入第($\alpha M+\sqrt{O((\log M)/M)}$)项所需要的预期数。

71.[*40*]　当把算法 C 修改成如正文中所描述的那样适合于外部查找时,试对它的特性做实验。

▶72.[*M28*](*万能散列*)想像一个巨大的矩阵 H,它对于每个可能的键码都有一个列。H 的元素被编号为 0 和 $M-1$ 之间的数。H 的行表示散列函数。如果 H 的任何两列至多在 R/M 行处相一致,其中 R 是行的总数,则说 H 定义了*散列函数的一个万能类*。

a)试证明,如果 H 是在这个意义下万能的,而且如果我们通过随机地选择 H 的一行来选择一个散列函数 h,则在分开拉链的这个方法(图 38)中包含任何给定的键码 K 的表列的预期大小,在我们插入 N 个不同的键码 $K_1,K_2,\cdots,K_N$ 的任何集合之后,将是 $\leqslant 1+N/M$。

b)假设(9)中的每个 h_j 是随机地选择的从所有字符的集合到集合 $\{0,1,\cdots,M-1\}$ 的映射,证明这对应于散列函数的万能类。

c)如果对于所有 j, $h_j(0)=0$,但对于 $x\neq 0$, $h_j(x)$ 是随机的,(b)的结果仍将成立吗?

73.[*M26*](Carter 和 Wegman)证明,即使当 h_j 不是完全随机的函数时,上题的部分(*b*)仍成立,但它们有下列特殊形式之一:(i)令 x_j 是二进数 $(b_{j(n-1)}\cdots b_{j1}b_{j0})_2$,则 $h_j(x_j)=(a_{j(n-1)}b_{j(n-1)}+\cdots+a_{j1}b_{j1}+a_{j0}b_{j0})\bmod M$,其中每个 a_{jk} 是随机地选择 modulo M 的。(ii)设 M 是质数,并假定 $0\leqslant x_j<M$,则 $h_j(x_j)=(a_jx_j+b_j)\bmod M$,其中 a_j 和 b_j 是随机地选择 modulo M 的。

74.[*M29*]　设 H 定义一个散列函数万能类。证明或反驳:给定任何 N 个不同的列,并且随机地选择任何行,则在这些列中 0 的预期数是 $O(1)$ 和 $O(N/M)$。[于是,在分开拉链方法中的每个表列都将有这个预期的大小。]

75.[*M26*]　证明或反驳关于(9)的散列函数的下列命题,当 h_j 是独立随机函数时:

a)对于所有 $0\leqslant m<M$, $h(K)=m$ 的概率是 $1/M$。

b)如果 $K\neq K'$,则对于所有 $0\leqslant m,m'<M$, $h(K)=m$ 和 $h(K')=m'$ 的概率是 $1/M^2$。

c)如果 K,K' 和 K'' 不相同,则对于所有 $0\leqslant m,m',m'',m'''<M$, $h(K)=m$, $h(K')=m'$ 和 $h(K'')=m''$ 的概率是 $1/M^3$。

d)如果 K,K',K'' 和 K''' 不相同,则对于所有 $0\leqslant m,m',m'',m'''<M$, $h(K)=m$, $h(K')=m'$, $h(K'')=m''$ 和 $h(K''')=m'''$ 的概率是 $1/M^4$。

76.[*M21*]　提出一个方法对于具有可变长的键码修改(9),并保持万能散列的性质。

77.[*M22*]　设 H 定义从 32 个二进位键码到 16 个二进位键码的散列函数的一个万能类(于是在习题 72 的记号下 H 有 2^{32} 个列, $M=2^{16}$)。一个 256 个二进位的键码可以看作是八个 32 个二进位部分 $x_1x_2x_3x_4x_5x_6x_7x_8$ 的连接;我们可以通过散列函数 $h_4(h_3(h_2(h_1(x_1)h_1(x_2))h_2(h_1(x_3)h_1(x_4)))h_3(h_2(h_1(x_5)h_1(x_6))h_2(h_1(x_7)h_1(x_8))))$,把它映射到一个 16 个二进位的地址,其中 h_1,h_2,h_3 和 h_4 是 H 的随机和独立地选择的行。(这里,例如 $h_1(x_1)h_1(x_2)$ 代表通过连接 $h_1(x_1)$ 和 $h_1(x_2)$ 得到的 32 个二进位的数。)试证明两个不同的键码散列到相同地址的概率小于 2^{-14}。[这个方案要求比(9)少很多的随机选择。]

She made a hash of the proper names, to be sure.

为了保险,她做了对于专用名字的一次散列。

——Grant Allen (*The Tents of Shem*, 1889)

6.5 利用辅助键码的查找

我们已经完成了对于主键码的查找,即唯一地确定文件中一个记录的键码查找的研究。但有时也有必要不是按主键码而是按记录中其它字段的值进行查找;这些其它的字段通常称为*辅助键码*或记录的*属性*。例如,在一个包含某大学学生信息的注册文件中,可能希望查找来自俄亥俄州不是专攻数学或统计学的二年级学生;或者查找所有未婚的说法语的研究生中的女生;等等。

一般我们假定,每个记录包含有若干属性,而我们要查找其某些属性具有某些值的所有记录。指定所需记录的说明称作一个*查询*。查询通常限于下列三种类型。

a)*简单查询*,它给出一个特定属性的一个特定的值;例如“MAJOR = MATHEM-ATICS”;或者“RESIDENCE. STATE = OHIO”。

b)*范围查询*,它给出一个特定属性值的一个特定的范围;例如,“COST < \$ 18.00”;或者“21 < AGE ≤ 23”。

c)*布尔查询*,它由与操作 AND,OR,NOT 相结合的上述类型组成;例如

“(CLASS = SOPHOMORE)AND(RESIDENCE·STATE = OHIO)
AND NOT((MAJOR = MATHEMATICS)OR(MAJOR = STATISTICS))”

对于这三类查询,发现有效查找技术的问题已经十分困难。因此,通常不再考虑更复杂类型的查询问题。例如,一个铁路公司可以有记载它的所有货车当前状态的一个文件;但不允许直接提像“找出在距西雅图 500 英里内所有空冷藏车”这类查询,除非“同西雅图的距离”是保存在每个记录内的一个属性而不是由其它属性导出的一个复杂函数。而且使用除 AND,OR 和 NOT 之外的逻辑量词,将引进仅仅受提出查询者的想像所限制的进一步的复杂性;例如,给定垒球统计的一个文件,我们可以问及在晚场比赛中最长的连续的击中表演。这些例子很复杂的,但它们仍可以通过扫描一个适当排列的文件来处理;其它的查询甚至更困难,例如,求在五个或更多的属性上有相同值的所有记录对(而不必确定哪一些属性必须匹配)。这样的查询可以认为属于一般程序设计的任务而超出了本书讨论的范围,但通常它们可被分成属于这里所考虑的问题类型的一些小问题。

在我们开始研究辅助键码检索的各种技术之前,重要的是考虑这个课题的经济效益。尽管大量的应用都能塞进上面概述的三类查询的一般框框里,但是这些应用中有许多并不真正适合于我们将要研究的复杂技术,而且它们中的某些用人工做比用机器做倒更好些!人们攀登珠穆朗玛峰,“因为它是客观存在”,而且因为已经发展了使得攀登成为可能的工具;类似地,当面对着数据的高山时,人们试图在一个联机实时环境中,利用一台计算机找出对他们所能想像到的最为困难的查询的回答,而无需顾及代价。所期望的计算是可能的,但未必对每一个人的应用都合适。

例如,考虑下列对于辅助键码检索的简单方法:在*成批化*一些查询之后,我们可以顺序地进行对整个文件的查找,检索出所有有关的记录(“成批化”指的是在处理

任何查询之前，先积累起一批这样的查询）。如果文件不太大并且这些查询不需要立即处理的话，这个方法是十分令人满意的。甚至对于磁带文件也可以使用，而且它仅仅在计算机有空间时才使用它，所以就设备代价来说，它往往非常经济。而且它甚至能处理上面讨论的“到西雅图的距离”这样的计算查询。

简化辅助键码检索的另一个简单方式，是提供给人们适当的打印好的索引信息，让他们自己做一部分工作。这通常是最合理和最经济的方式（当然，假定当打印新的索引时，旧的索引已被回炉）。特别是因为当人们能方便地访问大量数据时，他们倾向于去注意那些有趣的模式。

对某些应用来说，上述简单方案的处理不能令人满意，这些应用都涉及非常大型的文件，对于这样的文件，查询的快速回答很重要。例如，如果不断地有好几个用户同时查询文件，或者如果这些查询是机器而不是人所生成的，就会出现这种情况。在本节中，我们的目标是要了解在有关文件结构的各种假定之下，在通常的计算机上，对于辅助键码检索我们能做得多好。幸而，随着计算的费用继续显著地减少，我们将要讨论的方法在实用中正变得越来越可行。

为了处理这一问题，已经提出了大量好的想法，但是（如同读者将能从所有这些预先警告式的注释中猜测到的那样）这些方法决不意味着它们都像主键码检索算法那样好。由于文件及应用的多样性，我们不可能对已经考虑到的所有可能性给出完备的讨论，也不可能分析每个算法在典型环境中的特性。本节的剩下部分介绍已经提出的一些基本方法，至于在每一个具体情况下什么样的技术组合是最适当的，这个问题留给读者自己去想。

反文件 用于辅助键码检索的第一类重要的技术是*反文件*的思想。这并不意味着文件被上下颠倒，它指的是记录和属性的作用被逆转。我们不是列出一个给定文件的各属性，而是列出有一个给定属性的各记录。

在日常生活中，经常地遇到反文件（在其它名称之下）。例如，对应于俄英字典的反文件是一部英俄字典。对应于本书的反文件是附于本书末尾的索引。财会人员传统地使用“双项簿记”，其中所有的交易同时记入现金账和顾客账，于是当前的现金状况和顾客的债务都很容易查阅。

一般地说，一个反文件并不是独立的，它要同原来的未反过来的文件一起使用。它提供了重复的冗余信息以便加速辅助键码的检索。每个反文件由一些*反表列*组成，即其某个属性具有给定的值的所有记录的表。

正像所有的表一样，在一台计算机内的反表可以用许多方式来表示，而且不同的表示方式适用于不同的场合。某些辅助键码字段仅仅有两个值（例如，“性别”属性），而对应的反表却十分长；但是其它字段一般都有大量的值，而重复较少（例如，“电话号码”属性）。

例如，想像我们要在一本电话簿中存储信息，使得所有的项都可以根据名字、电话号码或居住地址进行检索。一个简单的解决办法是作成三个分开的文件，分别面向对于每种类型键码的检索。另一个思想是组合这些文件，例如可以建立三个散列

表,分别用作拉链方法的表头。在后一种方案中,文件的每个记录将是这三个表中的一个元素,因而它将包含三个链接字段;这就是在 2.2.6 小节图 13 中说明过了的和下面要进一步讨论的所谓*多重表*方法。第三种可能性是模拟图书馆卡片目录,在那里作者卡片、书名卡片以及主题卡片分别按字母顺序排在一起,用类似的方式可把三个文件组合成一个超级文件。

考察本书索引所用的格式,即可得到关于反表表示的进一步的思想。在有些辅助键码字段中,每个属性值一般有五个左右的项,此时我们只需作一个简短的顺序表,其中首先列出键码的值,后面跟以记录的位置(类似于一本书的索引中的页数)。如果有关的记录趋向于连续地堆积,则一个“范围说明”码(例如,页 521～542)是有用的。如果文件的记录经常要重新分配,则在反文件中使用主键码来代替辅助键码可能更好,使得当位置改变时也不必进行更改;例如,对于“圣经”段落的引用总是通过章和节给出,而对于某些书的索引都是基于节数而不是按页数给出的。

这些思想当中没有一个特别适合于像“性别”这样两个值的属性。在这种情况下,当然仅仅需要一个反表,因为非男性必然是女性而且反之亦然。如果每个值同大约一半的文件项有关,则反表将惊人地长,但是,在一台二进计算机上可以使用一种二进位串表示法,每个二进制位确定一个具体记录的值,即能相当好地解决这个问题。例如二进位串 01001011101…可能意味着文件中的头一个记录指的是一个男子,第二个记录指的是一个女子,下两个记录指的都是男子,等等。

这样的方法足以处理对特定属性值的简单查询。稍作推广即可以处理范围查询,不过必须用某个以比较为基础的查找方案(6.2 节)来代替散列。

对于像“(MAJOR = MATHEMATICS) AND (RESIDENCE. STATE = OHIO)”这样的布尔查询,我们需要把两个反文件交在一起。这可以用若干种方法来进行;例如,如果两个表都是有序的,则对每个表进行一次扫描将选出所有共同的项。或者,我们可以选出最短的表并考查它的每个记录,校验其它的属性;但这个方法仅对 AND 有效,而对 OR 无效,而且它对于外部文件没有吸引力,因为它要对许多不满足查询要求的记录进行访问。

同样的考虑表明,上面所描述的多重表组织,对于在一个外部文件上的布尔查询是低效的,因为它涉及许多不必要的访问。例如,想像如果本书的索引被组织成一个多重表的形式,则会发生什么情况。这意味着,索引的每个项都将仅仅指出提到该主题的最后一页;然后在每一页上,对于该页的每个主题,又有对于该主题的上一次出现的进一步索引。为了找出有关“[算法的分析]和[(外部排序)或(外部查找)]”的所有页面,必须翻动许多页。另一方面,通过只查看实际出现该索引的两个页,对反文件作简单的操作以便找出满足查询的页的最小子集,便可解决同样的查询问题。

当一个反表被表示作一个二进位串时,简单查询的布尔组合当然容易实现,因为计算机可以用相当高的速度对二进位串进行操作。在某些混合查询中,某些属性被表示作由记录号码排成的顺序表,而其它属性表示作二进位串,此时不难把顺序的表转换成二进位串,然后对这些二进位串实施布尔操作。

这里,给出一个假设的应用的定量例子,可能会有帮助。如同 5.4.9 小节所述,假定我们有每个为 40 个字符的 1 000 000 个记录,而且文件存储在 MIXTEC 磁盘上。文件本身因此填满两个磁盘组,反表大概将填满更多个磁盘组。每道包含 5000 个字符 = 30 000 个二进位,所以一个具体属性的反表至多花费 34 道(当二进位串的表示为最短时出现这个极大道数)。假设有一个稍微复杂些的查询,它涉及 10 个反表的一个布尔组合;在最坏的情况下,我们将要从反文件读 340 道的信息,总共花费 $340\times25\text{ms}=8.5\text{s}$ 的阅读时间。平均的等待延迟将大约是读时间的一半,但通过小心的程序设计有可能消除这个等待。通过把每个二进位串表列的第一个道存入一个柱面中,而每个表列的第二个道存于下一个柱面当中等,将消去大多数寻找时间,所以我们可以估计寻找时间至多大约是 $34\times26\text{ms}\approx0.9\text{s}$(如果包含有两个磁盘组,则是这数的两倍)。最后,如果 q 个记录满足查询,则对于每个记录,我们将花费大约 $q\times(60\text{ms}(\text{寻找})+12.5\text{ms}(\text{等待})+0.2\text{ms}(\text{读}))$ 的额外时间来取出它以便进行下一步处理。于是,为处理这个稍为复杂的查询,总计预期时间的乐观估计大约是 $<(10+0.073q)\text{s}$。这与在同样的假定下但不使用任何反表以最高速度读整个文件的 210s 时间形成对照。

这个例子表明在磁盘存储中空间的最优化同时间最优化是紧密相关的;处理反表的时间大致就是寻找和读它们所需的时间。

上面的讨论或多或少假定,当我们查询文件时,它不增长或收缩;如果需要经常更改,则应该作什么呢?在许多应用中,只需累积一批更改的要求,而且当不需要回答查询时,在空闲的时间里处理它们即可。或者,如果更改的文件优先级较高,则 B 树(6.2.4 小节)的方法是有吸引力的。反表的整个集合可被作成一个巨大的 B 树,同时对叶作特殊的约定使得分支节点包含键码的值,而诸叶既包含键码又包含记录指针的表。对文件的修改也可通过我们以下将要讨论的其它方法处理。

几何数据　大量的应用涉及在二维或更多维的空间中的点、线和形状。解决面向距离的最初的方法之一是由 Bruce McNutt 于 1972 年提出的“邮局树”。例如,假设给定 x 的值,我们希望来处理像“最靠近 x 的城市是哪一个?”这样的查询。McNutt 树的每个节点对应于一个城市 y 和一个“测试半径”r;这个节点的左子树对应于随继进入树的这个部分的所有城市 z,使得从 y 和 z 的距离 $\leqslant r+\delta$,类似地右子树是对于距离 $\geqslant r-\delta$ 的那些城市。这里 δ 是一个给定的容差。和 y 的距离在 $r-\delta$ 和 $r+\delta$ 之间的城市必须进入到两个子树中。在这样一株树中的查找使得找出在一个给定点的距离 δ 之内的所有城市成为可能(参见图 45)。

McNutt 和 Edward Pring 基于这个思想进行了若干次试验,并且以随机次序使用了美国大陆中 231 个人口最多的城市作为一个示例数据库。他们以有规则的方式使测试半径收缩,当向左时以 $0.67r$ 代替,而当向右时以 $0.57r$ 代替,但当取两个连续的右分支的第二个时 r 维持不变。结果是,在对于 $\delta=20$ 英里的树内,要求 610 个节点,而对于 $r=35$ 英里的树内要求 1600 个节点。图 45 示出他们最小的树的顶层几级。(在这株树剩下的一些级中佛罗里达的奥兰多出现在杰克逊维尔和迈

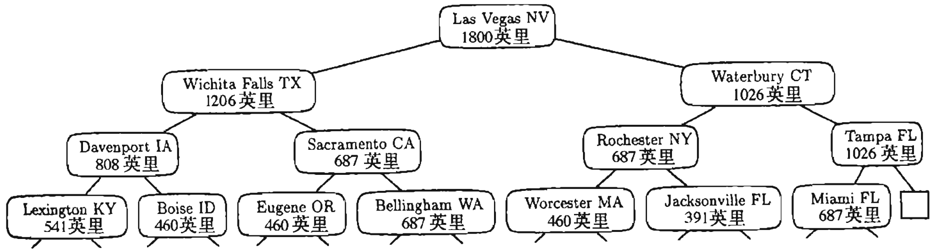


图 45 “邮局树”例子的顶部诸级。为了查找一个给定点 x 附近的所有城市，由根点开始:如果 x 在距离拉斯维加斯 1800 英里之内则向左,否则就向右;然后重复这过程直到遇到一个终端节点。树构造的方法确保在这个查找期间将会找到所有和 x 的距离在 20 英里之内的城市

阿密之下。某些城市十分频繁地出现;例如,这些节点中有 17 个都是马萨诸塞州的布洛克赖!)

随着 δ 的增加,文件的迅速增长表明,邮局树大概有有限的用途。通过直接地使用每个点的坐标,并把坐标当作属性或辅助键码,我们可以做得更好;然后我们可以基于键码的范围来作布尔查询。例如,假设文件记录涉及北美诸城市,而且查询问及满足

$$(21.49° \leqslant 纬度 \leqslant 37.41°)\text{AND}(70.34° \leqslant 经度 \leqslant 75.72°)$$

的那些城市。查阅地图可看出,许多城市满足这个纬度范围,必有许多城市满足经度范围,但是很难有任何城市处于这两个范围之中。解决这样的正交范围查询的一个方法是相当粗略地分划所有可能的纬度和经度的值,且每个属性仅仅有很小几类(例如,通过在下一个较小的 5°的倍数处截断),然后对于每一(纬度,经度)类组合有一个反表,这就好像有对于每个局部区域各一页的地图。利用 5°的区间,上边的查询将参考八个页,即(20°,70°),(25°,70°),…,(35°,75°)。现在需要对于这些页中的每一页处理范围查询,或者通过在这个页内进行一个更细的分划,或者通过直接查阅诸记录本身,采用哪种方法依赖于对应于该页的记录个数。在某种意义下,这是在每个内部节点处具有二维分支的一个树结构。

对于这个方法的一个实质性的精心改进,称作一个格子文件,是由 J. Nievergelt,H. Hinterberger 以及 K. C. Sercik 研制的[*ACM Trans. Database Systems* **9** (1984),38~71]。如果每个点 x 都有 k 个坐标 $(x_1,\cdots,x_k)$,他们把第 i 个坐标的值划分成以下的范围

$$-\infty = g_{i0} < g_{i1} < \cdots < g_{ir_i} = +\infty \tag{1}$$

并且通过确定下标 $(j_1,\cdots,j_k)$ 使得对于 $1 \leqslant i \leqslant k$,

$$0 \leqslant j_i < r_i, \quad g_{ij_i} \leqslant x_i < g_{i(j_i+1)} \tag{2}$$

来找出 x。有一个给定的$(j_1,\cdots,j_k)$的值的所有点称为单元。在相同单元内的点的记录存入外部存储的相同的桶当中。假定每个桶对应于一个 k 维的矩形区域或者"超单元",则也允许桶包含来自若干相邻的单元的点。可能有各种用于修改格子边界值 g_{ij} 和用于分开和合并桶的策略;例如,参见 K. Hinrichs,*BIT* **25** (1985),569～592;M. Regnier,*BIT* **25**(1985),335～357;P. Flajolet 和 C. Puech,*JACM* **33** (1986),371～407,§4.2 分析了带有随机数据的格子文件的特征。

使用所谓的*四叉树*结构,J. L. Bentley 和 R. A. Finkel 引进了处理正交范围查询的一个更简单的方法[*Acta Informatica* **4** (1974),1～9]。在他们构造的二维情况下,这样一株树的每个节点表示一个矩形,而且在该矩形中包含诸点之一。有四个子树,相对于给定点的坐标,他们对应于原来的矩形的四个像限。类似地,在三维中,有八路分支,而且有时把树叫做*八叉树*。一个 k 维的四叉树有 2^k 路分支。

对于随机四叉树的数学分析十分困难,但在 1988 年,由两组研究人员独立地工作,确定出在一个随机的 k 维四叉树中对于第 N 个节点的预期的插入时间的渐近形式是

$$\frac{2}{k}\ln N + O(1) \tag{3}$$

参见 L. Devroye 和 L. Laforest,*SICOMP* **19** (1990),821～832;P. Flajolet,G. Gonnet,C. Puech 以及 J. M. Robson,*Algorithmica* **10** (1993),473～500。注意当 $k=1$ 时,这个结果同对于一个二分查找树的插入的著名公式,即等式 6.2.2-(5)一致。P. Flajolet,G. Labelle,L. Laforest 和 B. Salvy 事实上证明了,平均内路径长度可以以令人惊讶地优美的形式

$$\sum_{l\geqslant 2}\binom{N}{l}(-1)^l\prod_{j=3}^{l}\left(1-\frac{2^k}{j^k}\right) \tag{4}$$

来加以表达,而且因此借助于超几何函数有可能对于随机四叉树作进一步的分析。[参见*Random Structures and Algorithms* **7** (1995),117～144]。

通过引进"k-d 树",Bentley 甚至更进一步地继续简化四叉树表示,这种树在每个节点处仅有两路分支。[*CACM* **18** (1975),509～517;*IEEE Transactions* **SE-5** (1979),333～340]。如同在 6.2.2 小节中那样,一个 1-d 树就是一个通常的二分查找树;一个 2-d 树也类似,但是当分支时在偶极上的节点比较 x 坐标,而在奇级上的节点比较 y 坐标。一般地说,一个 k-d 树有带 k 个坐标的节点,而且在每级上的分支仅仅基于坐标之一;例如,我们可以在级 l 上按坐标数$(l \bmod k)+1$ 来分支。可以使用基于记录的顺序号码或在内存中的位置的打破平衡规则以确保没有两个记录在任何坐标位置中一致。随机地增长的 k-d 树结果证实和通常的二分查找树有相同的平均路径长度和形状分布,因为构成它们增长基础的假定和在一维情况下是一样的(参见习题 6.2.2-6)。

如果文件不动态地变化,我们可以通过选择在每个节点进行分支的中间值,来平衡任何 N 节点的 k-d 树,使得它的高度$\approx\lg N$。于是我们可以确信,若干基本类型的查询都将被有效地处理。例如,Bentley 证明,我们可以在 $O(N^{1-t/k})$步内标识

出有 t 个确定的坐标的所有记录。如果坐标中的 t 个被限制为子范围而且总共有 q 个这样的记录,我们也可以在至多 $O(tN^{1-1/k}+q)$步内找出位于一个给定的矩形区域内的所有记录[李德财和黄泽权,*Acta Informatica* **23** (1977),23～29]。事实上,如果给定的区域接近是立体的而且 q 很小,以及如果在每个节点用于分支所选定的坐标有最大的散开的属性值,Friedman,Bentley 和 Finkel[*ACM Trans. Math. Software* **3** (1977),209～226]证明,对于这样一个区域查询的平均时间将仅仅是 $O(\log N+q)$。当在 k 维空间中对于一个给定的点最接近的邻域查找这样的 k-d 树时,同样的公式适用。

当 k-d 树是随机的而不是完美地平衡时,对于 t 个确定的坐标的部分匹配的平均运行时间稍微增加成 $\Theta(N^{1-t/k+f(t/k)}$;这里函数 f 是由以下方程

$$(f(x)+3-x)^x(f(x)+2-x)^{1-x}=2 \tag{5}$$

含蓄地定义的,因而它十分小;我们有

$$0\leqslant f(x)<0.06329\ 33881\ 23738\ 85718\ 14011\ 27797\ 33590\ 58170- \tag{6}$$

而且当 x 接近于 0.585 时出现极大值。[见 P. Flajolet 和 C. Puech,*JACM* **33** (1986),371～407,§3]。

由于几何算法的艺术魅力和巨大重要性,在解决高维查找问题以及许多类型的相关问题的技术方面有了巨大的增长。确实,自 20 世纪 70 年代以来,称为计算几何学的数学和计算机科学的新的子领域已经迅速地发展起来。由 J. E. Goodman 和 J. O'Rourke 主编的*The Handbook of Discrete and Computational Geometry* (佛罗里达州波卡拉顿:CRC 出版社,1997)是直到 1997 年为止关于该领域的技术发展现状的一本出色的参考书。

Hanan Samet 在两本互相补充的书*The Design and Analysis of Spatial Data Structure* 以及*Applications of Spatial Data Structures* (Addison-Wesley,1990)中,已经给出了关于二维和三维对象的重要特殊情况的数据结构和算法的广泛的综述。Samet 指出,现在把 Bentley 和 Finkel 原来的四叉树叫做"点四叉树"更适当。"四叉树"这个名称本身已经成为对于几何数据的任何层次分解的一个普通的术语。

复合属性 有可能把两个或多个属性组合成一个超级属性。例如,一个"(CLASS,MAJOR)属性"可由一个大学的注册文件的 CLASS 字段和 MAJOR 字段组合而成。这样以来,用不相交的短的表的并代替较长的表的交,常常可满足查询的需要。

林耀桑进一步发展了属性组合的思想[*CACM* **13** (1970),660～665],他建议从左到右地按字典顺序排列组合属性的反表,且拷贝许多份,并以一种巧妙方式排列组合中的各属性。例如,假设我们有三个属性 A,B,C;则可以形成三个复合属性

$$\texttt{(A,B,C)},\quad \texttt{(B,C,A)},\quad \texttt{(C,A,B)} \tag{7}$$

而且对于其中的每一个,构造有序的反表。(于是在第一个表中,诸记录以它们 A 值的顺序出现,而所有具相同 A 值的记录以 B 值的顺序,然后再按 C 值的顺序出现)。

这种组织有可能满足以三个属性的任何组合为基础的查询;例如,A 和 C 有特定值的所有记录都将连续地出现于第三个表中。

类似地,由四个属性 A,B,C,D,我们可以形成六个组合属性

$$(A,B,C,D),\quad (B,C,D,A),\quad (B,D,A,C),\quad (C,A,D,B),\quad (C,D,A,B),\quad (D,A,B,C) \tag{8}$$

它们足以回答有关这些属性中一个、两个、三个或四个值的简单查询的所有组合。有一个一般的过程可从 n 个属性构造出 $\binom{n}{k}$ 个组合属性,其中 $k \leqslant \frac{1}{2}n$,这使得有至多 k 个或至少 $n\text{-}k$ 个属性值的特定组合的所有记录,都将在组合的一个属性表中连续地出现(见习题 1)。或者,当某些属性只有有限个值时,我们只需较少的组合就可以了。例如,如果 D 只是一个二值的属性,则通过把 D 放在(7)的前边得到的三个组合

$$(D,A,B,C),\quad (D,B,C,A),\quad (D,C,A,B) \tag{9}$$

几乎将和(8)一样地好,而且只有一半的冗余,因为不依赖于 D 的诸查询,可以只通过考察这些表之一的两个位置来进行处理。

二进属性　考虑所有属性都是二值的这一特殊情况是有教益的。在某种意义下,这是把属性组合起来的*对立面*,因为我们可以把任何值表示成为二进数,而且把这个数的个别的二进位认为是分开的属性。表 1 示出了涉及"是否"属性的一个典型文件;在这个情况下,诸记录代表被选定的甜饼的食谱,而诸属性确定使用哪些成分。例如,杏仁风味薄脆饼是用黄油、面粉、牛奶、坚果仁以及颗粒糖制造的。如果我们把表 1 想像作 0 和 1 的矩阵,则该矩阵的转置,就是在二进位串形式下的"反"文件。

表 1 的右边一列用来指出很少出现的特殊项目。比起为每个项目都设一列来,这样的方式可以更有效地进行编码;"玉米淀粉"这一列可以类似地处理。对偶地,我们可以发现对于"面粉"这一列进行编码的更有效的方式,因为面粉在除蛋白甜饼之外的每一行当中都出现。但现在我们把这些考虑放在一边,而简单地忽略"特殊成分"一栏。

让我们定义二进属性文件中的一个*基本查询*为:对于在某些列中值为 0 和在其它列中值为 1,以及在剩下的列中值为任意的所有记录的一个请求。利用"*"来代表一个任意的值,我们可以把任何基本查询表示为诸 0,1 和 * 的序列。例如,考虑喜欢某种椰子饼干,但却厌恶巧克力,讨厌茴香,以及不喜欢提纯香精的人;他可以描述查询如下

$$*0****0**1*********************0 \tag{10}$$

现在表 1 指出甜美条恰是这种东西。

在考虑为基本查询而组织文件的一般问题之前,先来考察没有指定 0 值,而仅仅指定 1 值和 * 值这种重要的特殊情况。如果我们假定 1 表示存在的属性,而 0 表示不存在的属性,则这可以叫做一个*包含查询*,因为它问及包括某个属性集合的所

表 1　具有二项属性的一个文件

	多香果	茴香籽	发酵粉	发酵苏打	黄油	小豆蔻	巧克力	桂皮	丁香	椰子	咖啡	玉米淀粉	枣	蛋白	蛋黄	面粉	姜汁	柠檬汁	柠檬皮	牛奶	糖浆	肉豆蔻	坚果仁	燕麦片	葡萄干	盐	红糖	颗粒糖	面糖	提纯香精	特殊成分
杏仁风味薄脆饼	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	1	0	0	—
苹果酱香味方饼	0	0	0	1	1	0	0	1	1	0	0	0	0	1	1	1	0	0	0	0	0	1	1	0	1	0	0	1	1	1	苹果酱
香蕉燕麦片家常小甜饼	0	0	0	1	1	0	0	1	0	0	0	0	0	1	1	1	0	0	0	0	0	1	1	1	0	1	0	1	0	1	香蕉
巧克力薄片小甜饼	0	0	0	1	1	0	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	1	0	1	0	1	—
椰子杏仁饼	0	0	1	0	1	0	0	0	0	1	0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	1	0	1	0	1	—
奶油乳酪小甜饼	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	奶油乳酪
美味梅脯条饼	0	0	0	0	1	0	0	0	0	1	0	1	0	1	1	1	0	0	0	0	0	0	1	0	0	1	1	0	0	0	桔子梅脯
双料巧克力糖饼	0	0	0	1	1	0	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	1	0	1	0	0	—
甜美条饼	0	0	1	0	1	0	0	0	0	1	0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	1	1	0	0	1	—
带馅半圆形卷饼	0	0	1	0	1	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	0	0	1	0	0	1	0	1	0	1	—
芬斯卡·卡客(一种食品名)	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	1	0	1	0	0	提纯杏仁
浇糖姜汁饼	0	0	1	1	1	0	0	1	1	0	0	0	0	1	1	1	1	0	0	1	1	0	0	0	0	0	1	0	1	0	醋
胡桃葡萄干甜饼	0	0	0	1	1	0	0	1	0	0	1	0	0	1	1	1	0	0	0	0	0	1	0	0	1	1	1	0	0	0	杏
宝石家常甜饼	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	0	1	0	0	1	葡萄肉冻
环形甜薄饼	0	0	0	0	1	0	0	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	0	1	0	1	色拉油
波纹套圈饼	0	0	0	0	1	1	0	0	0	0	0	0	0	1	1	1	0	0	1	0	0	0	1	0	0	0	0	1	0	0	—
圆腿肉面包	1	0	0	1	0	0	0	1	1	0	0	0	0	1	1	1	0	1	1	1	0	1	1	0	0	1	1	0	1	0	蜂蜜
蛋白甜饼	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	蜜饯樱桃
摩拉维亚香味家常甜饼	1	0	0	1	1	0	0	1	1	0	0	0	0	1	0	1	1	0	0	0	1	1	0	0	0	1	1	0	1	0	—
燕麦枣椰子条饼	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	1	1	0	1	1	1	0	0	—
旧式家常糖饼	0	0	1	1	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	1	0	0	0	1	0	1	0	1	酸奶
花生黄油风车饼	0	0	0	1	1	0	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	1	花生黄油
裙状燕尾饼	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	1	—
佩弗纽斯	1	1	1	0	0	1	0	1	1	0	1	0	0	1	1	1	0	0	1	0	0	1	1	0	0	1	1	0	1	0	香橼,肉豆蔻干皮,胡椒
苏格兰燕麦脆饼	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	0	1	0	1	—
星状脆饼	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	—
春饼	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	1	0	0	0	0	0	0	1	0	1	1	0	—
斯普里兹家常甜饼	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	1	0	1	—
瑞典套圈饼	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	—
瑞士桂皮香脆片饼	0	0	0	0	1	0	0	1	0	0	0	0	0	1	1	1	0	0	0	1	0	0	0	0	0	1	1	1	0	0	—
太妃条	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	1	0	0	1	—
香精果仁冰镇家常饼	0	0	1	0	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	1	0	1	0	1	—

参考:McCall's Cook Book(New York:Random House,1963)第 9 章

有记录。例如，在表 1 的食谱中，既要求发酵粉又要求发酵苏打的，是浇糖姜汁饼和旧式家常糖饼。

在某些应用中，提供包含查询这种特殊功能就足够了。例如，在许多人工卡片文件系统的情况下就出现这种情形，诸如"带槽口边的卡片"或"特性卡片"。一个对应于表 1 的带槽口边卡片系统，对于每种花色将有一张卡片，而且对于每一成分都有穿出的孔(见图 46)。为了处理一个包含查询，卡片文件排成一叠而穿针放置在每个列中对应于所需属性的位置上。在提起穿针之后，有适当属性的所有卡片都将落下。

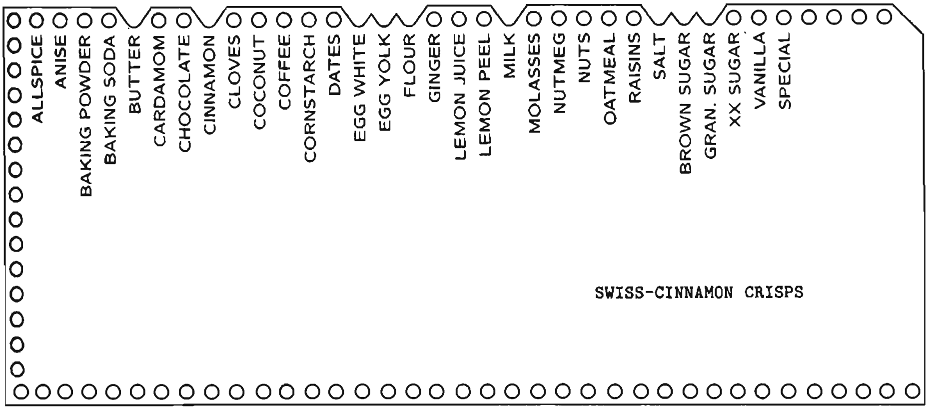


图 46 一张带槽口边的卡片

一个特性卡片系统以类似的方式对反文件进行工作。在这种情况下，每个属性有一张卡片，对于具有该属性的每个记录，孔就穿在这张卡片的指定位置上。因此，一张通常的 80 列卡片可用来告知 $12\times80=960$ 个记录中，哪一个有给定的属性。为了处理一个包含查询，先选出特定属性的特性卡片并把它们放在一起；然后，光线穿过对应于所要的记录的所有位置。这个操作类似于上面所说明的求颠倒的二进位串之交来处理布尔查询的情况。

叠加的编码 我们对于这些人工卡片系统有特殊兴趣的原因在于已经想出了一些精巧的方案来节省带槽口边的卡片上的空间；同样的原理可应用于计算机文件的表示上。叠加的编码是一项类似于散列的技术，而且在发现散列之前若干年，实际上它就被发明了。其思想是把诸属性映射到一个 n 位字段的随机 k 位代码中，并且对于记录中所出现的每一个属性的编码进行叠加。对某个属性集合进行的包含查询可以被转换成对于对应的叠加的二进位码的包含查询。额外的少量记录可能满足这个查询，但是这种"假情报"的个数在统计上是可以控制的[参考 Calvin N. Mooers, *Amer. Chem. Soc. Meeting* **112** (1947 年 9 月), 14E-15E; *American Documentation* **2** (1951), 20～32]。

作为叠加编码的一个例子，我们再次考虑表 1，但是不考虑像发酵粉、松酥油

脂、蛋、面粉这样的基本成分，而仅仅考虑调味品。表 2 表明，如果我们对于每一个调味品的属性都指定 10 个二进位字段中的 2 位随机代码，并叠加这些编码，将发生什么情况。例如，巧克力薄片小甜饼的项目，是通过叠加巧克力、坚果仁和香精的代码而得到的：

$$0010001000 \vee 0000100100 \vee 0000001001 = 0010101101$$

这些代码的叠加，也产生了某些假的属性，在这种情况下即多香果，蜜饯樱桃，葡萄肉冻，花生黄油，以及胡椒；这些将引起在某些查询中出现假情报。（而且它们也提出了建立一个称为假情报家常饼的新品种！）

叠加的编码实际上在表 2 中不能非常有效地工作，因为该表是有着大量属性存在的一个小例子。事实上，苹果酱香味方饼对于每个查询都将是一个假情报，因为它是通过叠加 7 个代码得到的，而结果是覆盖了全部十个位置；而圣诞小酥饼甚至更坏，它通过叠加 12 个代码得到！另一方面，表 2 在某些方面却工作得惊人地好；例如，在查询"提纯香精"时，只有圣诞小酥饼的记录作为假情报出现。

如果我们有，比如说，一个 32 位的字段以及 $\binom{32}{3}=4960$ 种不同属性的组合，其中允许每个记录具有多达六个属性，而且每个属性通过指定 32 个二进位中的 3 位来编码，则出现更适合的叠加编码的例子。在这种情况下，如果假定每个记录有六个随机选择的属性，则在一个包含查询中假情报的概率

在一个属性上是	.07948358	
在两个属性上是	.00708659	
在三个属性上是	.00067094	(11)
在四个属性上是	.00006786	
在五个属性上是	.00000728	
在六个属性上是	.00000082	

表 2　叠加编码的一个例子

个别调味品的代码			
提纯杏仁	0100000001	枣	1000000100
多香果	0000100001	姜　汁	0000110000
茴香籽	0000011000	蜂　蜜	0000000011
苹果酱	0010010000	柠檬汁	1000100000
杏	1000010000	柠檬皮	0011000000
香　蕉	0000100010	肉豆蔻干皮	0000010100
蜜饯樱桃	0000101000	糖　浆	1001000000
小豆蔻	1000000001	肉豆蔻	0000010010
巧克力	0010001000	坚果仁	0000100100
桂　皮	1000000010	桔　子	0100000100
香　橼	0100000010	花生黄油	0000000101

（续）

丁　香	0001100000	胡　椒	0010000100
椰　子	0001010000	梅　脯	0010000010
咖　啡	0001000100	葡萄干	010100000
葡萄肉冻	0010000001	提纯香精	0000001001
		叠加的代码	
杏仁风味薄脆饼	0000100100	圆腿肉面包	1011110111
苹果酱香味方饼	1111111111	蛋白甜饼	1000101100
香蕉燕麦片家常小甜饼	1000111111	摩拉维亚香味家常甜饼	1001110011
巧克力薄片小甜饼	0010101101	燕麦枣椰子条饼	1000100100
椰子杏仁饼	0001111101	旧式家常糖饼	0000011011
奶油乳酪小甜饼	0010001001	花生黄油风车饼	0010001101
美味梅脯条饼	0111110110	裙状燕尾饼	0000001001
双料巧克力糖饼	0010101100	佩佛纽斯	1111111111
奶油条饼	0001111101	苏格兰燕麦脆饼	0000001001
带馅半圆形卷饼	1011101101	星状脆饼	0000000000
芬斯卡·卡客饼	0100100101	春饼	0011011000
浇糖姜汁饼	1001110010	斯普里兹甜饼	0000001001
胡桃葡萄干甜饼	1101010110	瑞典套圈饼	0000000000
宝石家常甜饼	0010101101	瑞士桂皮香脆饼	1000000010
环形甜薄饼	1000001011	太妃条饼	0010111101
波纹套圈饼	1011100101	香精果仁冰镇家常饼	0000101101

于是，如果有 M 个不真正满足两属性查询的记录，则其中大约有 $.007M$ 个叠加的代码将虚假地符合两个指定属性的所有代码位（这些概率在习题 4 中计算）。在反文件中所需的总的二进位仅仅是记录数的 32 倍，不到在原来的文件中确定属性所需要的二进位数的一半。

如果使用小心地选择的非随机代码，则如同 W. H. Kautz 和 R. C. Singleton 所证明的，有可能使用叠加代码而不出现任何假情报，见*IEEE Trans*，**IT-10** (1964)，363～377；关于它们的构造之一，见习题 16。

Malcolm C. Harrison[*CACM* **14** (1971)，777～779]已经发现，叠加的编码可被用来加速文本查找。假定我们要在一个长的文本中找出一个特定字符串的所有出现，又不用像在算法 6.3P 那样构造很大的表；而且假定，该文本被分成为例如每个有 50 个字符的行 $c_1c_2\cdots c_{50}$。Harrison 建议通过把 49 个对偶 $c_1c_2, c_2c_3, \cdots, c_{49}c_{50}$ 中的每一个，比方说，散列成为 0～127 之间的一个数；则行 $c_1c_2\cdots c_{50}$的“标记”是

128 个二进位 $b_0b_1\cdots b_{127}$的串,其中 $b_j=1$ 当且仅当对某个 j,$h(c_jc_{j+1})=i$。

如果我们现在要在一个称为 HAYSTACK 的大型文件中查找字 NEEDLE 的所有出现,则只需考察所有其标记在 $h(NE)$,$h(EE)$,$h(ED)$,$h(DL)$和 $h(LE)$的位置上包含二进位 1 的行。假定散列函数是随机的,一个随机行在它的标记中包含所有这些二进位的概率仅仅是 0.00341(参考习题 4);因此,五个反表的二进位串的交将迅速查出包含 NEEDLE 的所有行,连同少量的假情报在一起。

随机性的假定在这个应用中并不非常有道理,因为典型的文件有如此多的冗余性;在英语词汇中,相邻字母对的分布是高度有偏向性的。例如,拼弃包含有一个空白字符的所有对偶 c_jc_{j+1}大概将是非常有帮助的,因为空白通常比任何其它符号更为常见得多。

Burton H.Bloom 已经提出了叠加编码对于查找问题的另一个有趣的应用[*CACM* **13** (1970),422～426];他的方法实际上适用于主键码的检索,然而在本节中讨论它却是最适当的。想像一个大型数据库的查找,而且在这个大型数据库中,如果查找是不成功的,就不需要进行计算。例如,我们有可能需要校验某人的信贷定额或护照号码等等,如果此人的记录在文件中没有出现,则无需作进一步的研究。类似地,在用计算机排字时,我们可以有一个简单的算法,它正确地连接大多数的字,但对大约 50 000 个例外的字它是失败的;如果我们没有发现例外文件中的字,就可以放心地使用这个简单的算法。

在这样一些情况下,有可能在内存中保持一个二进位表,使得在文件中不出现的大多数键码可以被认为是不存在的,而无需进行对外部存储的任何访问。做法是:设内部二进位表是 $b_0b_1\cdots b_{M-1}$,其中 M 相当大,对于文件中的每个键码 K_j,计算 k 个独立的散列函数$h_1(K_j)$,$\cdots h_k(K_j)$,并且置相应的 k 个 b 成为 1(这 k 个值不必是不同的)。于是当且仅当对于某一个 j 和l,$h_l(K_j)=i$ 时$b_i=1$。现在来确定一个查找变元 K 是否在外部文件中,首先测试对于 $1\leqslant l\leqslant k$,是否有 $b_{h_l}(K)=1$;如果不是,则不需要访问外存,但如果是,则若 k 和M 已适当地选择,一个通常的查找大概将找出 K。当在文件中有 N 个记录时,出现一个假情报的机会近似地为$(1-e^{-kN/M})^k$。在某种意义下,Bloom 的方法把整个文件处理成为一个记录,以主键码作为存在的属性,而且在一个庞大的 M-个二进位的字段中存放叠加编码。

Richard A. Gustafson 已经发展了叠加编码的另一种变形[Univ. South Carolina,1969]。假设我们有 N 个记录,而且每个记录具有从10 000 种可能的属性的集合中选出的六个属性。例如,这些记录可以代表技术论文,而属性可以是描述这篇文章的关键词。设 h 是一个散列函数,它把每个属性映射到 0 和 15 之间的一个数。如果一个记录有属性 $a_1,a_2,\cdots,a_6$,则 Gustafson 建议把这个记录映射成为 16 个二进位的数 $b_0b_1\cdots b_{15}$,其中 $b_i=1$ 当且仅当对某个 j,$h(a_j)=i$;而且,如果这个方法仅导致诸 b 中的k 个等于 1,这里 $k<6$,则另外的 $6-k$ 个 1 由某种随机方法提供之(不必依赖于记录本身)。在 $\binom{16}{6}=8008$ 种十六个二进位的代码中恰有六个 1 出

现,而且碰巧大约有 $N/8008$ 个记录将被映射到每个值上。我们可以保持诸记录的 8008 个表,利用一个适当的公式直接计算对应于 $b_0b_1\cdots b_{15}$的地址。事实上,如果在位置 $0\leqslant p_1<p_2<\cdots<p_6$ 中出现 1,则函数

$$\binom{p_1}{1}+\binom{p_2}{2}+\cdots+\binom{p_6}{6}$$

将把每个串 $b_0b_1\cdots b_{15}$转换成数 0 和 8007 之间的惟一的数,我们在习题 1.2.6-56 和 2.2.6-7 中已经看到了这一点。

现在,如果要找出具有三个具体属性 A_1,A_2,A_3 的所有记录,则我们计算 $h(A_1),h(A_2),h(A_3)$;假设这三个值是不同的,我们仅仅需要考察存储在$\binom{13}{3}=$ 286 个表中,且其二进制代码 $b_0b_1\cdots b_{15}$在这三个位置中包含 1 的诸记录。换言之,在查找中仅仅需要考察 286/8008≈3.5% 的记录。

关于叠加编码的一个精彩的讲述,连同它对于电话号码目录表的大型数据库的应用,参见 C. S. Roberts,*Proc IEEE* **67** (1979),1624～1642。J. K. Mullin 和 D. J. Margoliash,*Software Practice & Exper.* **20** (1990),625～630 讨论了它在检查拼写软件中的应用。

组合散列　奠定刚刚描述的 Gustafson 方法的思想是找出某种方法把记录映射到内存单元,使得仅有较少的单元同一个具体查询相关联。但是当各记录只有较少的属性时,他的方法仅可应用于包含查询。另一类映射设计来处理像(10)那样由诸 0,1 和 * 组成的任意基本查询,它是由 Ronald. L. Rivest 于 1971 年发现的[见 *SICOMP* **5** (1976),19～50]。

首先,假定我们要编写对于所有具有六个字母的英文字的一部交叉字谜字典,一个典型的查询可以是,比如说,查问所有具有形式 N * * D * E 的字,而得到的回答是{NEEDLE,NIDDLE,NODDLE,NOODLE,NUDDLE}。这个问题可巧妙地解决如下:建立 2^{12}个表,把字 NEEDLE 翻成表号

$$h(N)h(E)h(E)h(D)h(L)h(E)$$

这里 h 是一个散列函数,它把每个字母变成一个二进位值。把 6 个二进位对放在一起,即得一 12 位的表地址,因此,为了回答查询 N * * D * E,仅需查看 4096 张表中的 64 张表。

类似地假设我们有 1 000 000 个记录,每个记录含 10 个辅助键码,其中每个辅助键码有相当量的可能的值。我们可以把辅助键码为$(K_1,K_2,\cdots,K_{10})$的诸记录映射到 20 位的数

$$h(K_1)h(K_2)\cdots h(K_{10}) \tag{12}$$

上,其中 h 是把每个辅助键码映成 2 位值的散列函数,(12)表示这 10 个二进位对偶的并列。这个方案把 1 000 000 个记录映射成为 $2^{20}=1\ 048\ 576$ 个可能的值,而且可以把总共的映射当作具有 $M=2^{20}$的一个散列函数;拉链可用来解决冲突。如果我们要检索其任何五个辅助键码具有特定值的所有记录,则仅需要考察 2^{10}个表,对

应于(12)中的五个未确定的二进位对偶;所以平均仅仅需要考察大约 $1000=\sqrt{N}$个记录。(一个类似的方法已由 M. Arisawa, *J. Inf. Proc. Soc. Japan* **12** (1971), 163～167 和 B. Dwyer(未发表)提出。Dwyer 提出使用比(6)更灵活的映像,就是

$$(h_1(K_1)+h_2(K_2)+\cdots+h_{10}(K_{10}))\cdot\bmod M$$

其中 M 是任何合适的数,而 h_i 可能是 w_iK_i 形式的任意散列函数,其中 w_i 是"随机"的。

Rivest 已经进一步发展了这个思想,使得在许多情况下,我们有下列的状况。假定有 $N\approx2^n$ 个记录,每个有 m 个辅助键码。每个记录以这样一种方式映射到一个 n 位的散列地址上,即任何不指定 k 个键码的值的查询近似地对应于$N^{k/m}$个散列地址。在本节中(Gustafson 的方法除外),我们已经讨论过的所有其它的方法,都需要用阶为 N 的步骤进行检索,尽管比例常数很小;对于足够大的 N,Rivest 的方法将更快,而且它不需要反文件。

但是在我们能应用这项技术之前,要定义一个适当的映像。这里是具有小参数的一个例子,当 $m=4$ 和 $n=3$ 以及所有辅助键码的值都是二进数时;我们可以把 4 位记录映射成为八个地址如下:

$$\begin{array}{ll} *\ 0\ 0\ 1\rightarrow 0 & *\ 1\ 1\ 0\rightarrow 4\\ 0\ *\ 0\ 0\rightarrow 1 & 1\ *\ 1\ 1\rightarrow 5\\ 1\ 0\ *\ 0\rightarrow 2 & 0\ 1\ *\ 1\rightarrow 6\\ 1\ 1\ 0\ *\rightarrow 3 & 0\ 0\ 1\ *\rightarrow 7\end{array}\qquad(13)$$

对这个表的考察揭示,对应于查询 0 * * * 的所有记录,都映射到单元 1,2,5,7 和 8*;而且类似地,任何具有三个 * 的基本的查询恰恰对应到五个单元;具有两个 * 的基本查询每个对应于三个单元;而具有一个 * 的基本查询对应于一个或两个单元,平均为$(8\times1+24\times2)/32=1.75$,于是有

查询中未确定的二进位数	要查找的地址个数
4	$8=8^{4/4}$
3	$5\approx8^{3/4}$
2	$3\approx8^{2/4}$
1	$1.75\approx8^{1/4}$
0	$1\approx8^{0/4}$

(14)

当然,这是一个小例子,通过硬算更容易处理它。但它导致非显然的应用,因为当 $m=4r$ 和$n=3r$ 时我们也可以应用它:通过把键码分成每个 4 位的 r 个组,而且对每组应用(13),把 $4r$ 位的记录映射成为 $2^{3r}\approx N$ 个地址。得到的映射有所希望的性质:*在 m 个二进位中对 k 个不加指定的一个查询,将近似地对应到* $N^{k/m}$*个单元*(见习题 6)。

1997 年,A. E. Brouwer[*SICOMP* **28** (1999), 1970～1971]通过类似于(13)的一

* 此处原著有误,原文为 1,2,5,7 和 8,但地址中并无 8。——译者

个映射，发现把 8 个二进位压缩成 5 个二进位的一个有吸引力的方法。每个 8 个二进位的字节恰好属于下列 32 个类之一。

```
0*000*0*   01*0**11   00*11**1   *11**101
1*000*0*   11*0**11   10*11**1   *11**010
0*010*0*   01*1**11   00*0*01*   *10*0*10
1*010*0*   11*1**11   10*0*01*   *10*1*01
0*10*1*0   0*1*000*   *01*01*1   *0*1001*      (15)
1*10*1*0   1*1*000*   *10*10*0   *0*0100*
0*11*1*0   0*0*11*0   *00*011*   *0*011*1
1*11*1*0   1*0*11*0   *11*100*   *0*110*0
```

在这个设计中的 * 号是以下列方式安排的，即在每行中恰有 3 个，而在每列中恰有 12 个。习题 18 说明如何得到类似的方案。它将把有比如说 $m=4^r$ 个二进位的记录压缩成有 $n=3^r$ 个二进位的地址。在实用中，将使用大小为 b 的桶，而且我们将取 $N\approx 2^n b$；为了讲述的简单性，在上边的讨论中已经使用 $b=1$ 的情况。

Rivest 还提出了用于处理基本查询的另一个简单的技术。假设我们有，比如说，每个有 30 个二进位的 $N\approx 2^{10}$ 个记录，这里我们希望回答像(10)那样任意的 30 个二进位的基本查询。于是可以简单地把这 30 个二进位分成 3 个 10 个二进位的字段，并保持 3 个分开的大小为 $M=2^{10}$ 的散列表。每个记录分别存储 3 次，即在对应于它在 3 个字段中的二进位配置的各表列中。在适当条件下，每个表列将包含大约一个元素。给定一个带有 k 个未指定二进位的基本查询，则至少这些字段之一将有$\lfloor k/3\rfloor$或更少的二进位是未指定的；因此我们至多需要考察 $2^{\lfloor k/3\rfloor}\approx N^{k/30}$ 个表列，以找出对于这一查询的所有回答。或者，我们可以在已选择的字段中使用用于处理基本查询的任何其它技术。

广义的检索结构 基于类似于 6.3 节的检索结构这样一个数据结构，Rivest 继续提出另一个方法。我们可以令一个广义的二分检索结构的每个内节点确定它表示记录的哪个二进位。例如在表 1 的数据中，可以让检索结构的根表示香草精；于是左子检索结构将对应于不含香草精的 16 个食物品种，而右子检索结构将表示使用香草精的 15 个食物品种。这 16-15 的划分正好把文件分成两半；我们以类似的方式处理每个子文件。当一个子文件变成适当小时，即以一个终端节点表示它。

为处理一个基本的查询，我们从检索结构的根开始。当查找其根确定一个属性的一个广义检索结构时，若查询有 0 或 1 的值，我们就而分别地查找左或右子检索结构；如果该查询在该二进位置有 *，则两个子检索结构都查找。

假设属性不是二进制的，但以二进制来表示它们。我们可以通过首先考察属性 1 的头一个二进位，然后考察属性 2 的头一个二进位，……，属性 m 的头一个二进位，然后属性 1 的第二个二进位，等等，来构造一个检索结构。通过与 m-d 树的类比，这样一个结构叫做“m-d 检索结构(它通过比较而不是通过对位的检查来分支)。P. Flajolet和C. Puech已经证明，当属性中有 k/m 个未被指定时，为了回答在N个

节点的一个随机 m-d 检索结构中的一个部分匹配查询的平均时间是 $\Theta(N^{k/m})$，[*JACM* **33** (1986)，371～407，§4.1]；W. Schachinger 已经计算了方差，见*Random structures and Algorithms*，**7** (1995)，81～95。

对于数字查找树的 m 维变形和 6.3 节的 Patricia 树，可以设计类似的算法。这些结构，它们稍微趋向于比 m-d 检索结构更好地平衡，已经由 P. Kirschenhofer 和 H. Prodinger 作了分析，见*Random Structures and Algorithms* **5**(1994)，123～134。

* **平衡的文件方案** 对于信息检索的另一个组合方法，是以平衡的不完备的区组设计为基础的，这是进行过相当大量研究的课题。尽管这一课题从数学的观点看来是非常有趣的，但不幸的是，与上面描述的其它方法比较，还不能证明它是非常有用的。这里将给出简短的理论介绍，以便指出结果的特色，希望某些读者可以想出一个好方法，使这些思想变成为实用的。

Steiner *三元系统*是把 v 个对象排列成无次序的三元组，方式是每一对对象恰在一个三元组中出现。例如，当 $v=7$ 时，实际上仅有一个 Steiner 三元系统，即

三元组	包括的数对	
{1,2,4}	{1,2}，{1,4}，{2,4}	
{2,3,5}	{2,3}，{2,5}，{3,5}	
{3,4,6}	{3,4}，{3,6}，{4,6}	
{4,5,0}	{0,4}，{0,5}，{4,5}	(16)
{5,6,1}	{1,5}，{1,6}，{5,6}	
{6,0,2}	{0,2}，{0,6}，{2,6}	
{0,1,3}	{0,1}，{0,3}，{1,3}	

由于有 $\frac{1}{2}v(v-1)$ 对对象，且每个三元组有三个对，总共必然有 $\frac{1}{6}v(v-1)$ 个三元组；而且由于每个对象必须同 $v-1$ 个其它的对象配对，每一对象必然恰恰出现于 $\frac{1}{2}(v-1)$ 个三元组中。这些条件意味着除非 $\frac{1}{6}v(v-1)$ 和 $\frac{1}{2}(v-1)$ 都是整数，否则 Steiner 三元系统就不能存在，这等价于说 v 是奇数而且不同于 2 modulo 3，即

$$v \bmod 6 = 1 \text{ 或 } 3 \qquad (17)$$

反之，T. P. Kirkman 于 1847 年证明，Steiner 三元系统对所有使得(17)成立的 $v\geqslant1$ 存在。他的有趣的构造在习题 10 中给出。

Steiner 的三元组系统可用来减少组合的反文件索引的冗余性。例如，再次考虑表 1 家常饼食谱的文件，并把最右边的一列转换成第 31 个属性，当需要任何特殊成分时为 1 否则为 0。假定我们要回答关于属性对偶的所有包含查询，例如，"什么花

色既使用椰子又使用葡萄干?”对于这$\binom{31}{2}=465$种可能的查询之每一种,都可以建立一个反表。但结果将花费大量的空间,因为(比如说)圣诞小酥饼将在$\binom{17}{2}=136$个表中出现,而且具有所有31种属性的一个记录将在每个表中出现!在这种情况下一个Steiner三元组系统可用来作稍许改进。有一个31个对象的Steiner三元组系统,且有155个三元组,而且每个对象对偶恰恰出现于这些三元组之一中。我们可以为每个三元组$\{a,b,c\}$配备四个表,在这四个表中,一个是对于有属性$a,b,\bar{c}$(即a且b但非c)的所有记录的,另一个表是对于有属性$a,\bar{b},c$的,第三个是对于$\bar{a},b,c$,第四个是对于有所有三个属性a,b,c的。这就保证任何记录都不会被包括在155个以上的反表中,而且每当一个记录有对应于系统的某个三元组的三个属性时,它就节省了空间。

三元组系统是区组设计的特殊情况,其中的区组有三个或更多的对象。例如,有一个方法排列31个对象成为六元组,使得每对对象恰恰出现于一个六元组中。

$$\{0,4,16,21,22,24\},\{1,5,17,22,23,25\},\cdots,\{30,3,15,20,21,23\} \qquad (18)$$

(这个设计通过mod 31加法由第一个区组形成。为了验证它有所述的性质,注意对于$i\neq j$,30个值$(a_i-a_j)\bmod 31$是不同的。其中$(a_1,a_2,\cdots,a_6)=(0,4,16,21,22,24)$。为了求出包含给定对$(x,y)$的六元组,选择$i$和$j$使得$a_i-a_j\equiv x-y\pmod{31}$;现在,如果$k=(x-a_1)\bmod 31$,则我们有$(a_i+k)\bmod 31=x$和$(a_j+k)\bmod 31=y$)。

我们可以利用上述设计存储反表,使得没有任何一个记录出现31次以上。对于具有a,b,c,d,e,f诸属性中两个或两个以上属性的记录的各种可能性,每个六元组$\{a,b,c,d,e,f\}$同57个表相关联,即$(a,b,\bar{c},\bar{d},\bar{e},\bar{f})$,$(a,\bar{b},c,\bar{d},\bar{e},\bar{f})$,$\cdots$,$(a,b,c,d,e,f)$;而且对于每个2-属性包含查询的答案是适当的六元组的16个适当的表的析取。对于这个设计,圣诞小酥饼将存于31个区组的29个中,因为如果我们把列编号成0～30,则该记录在除了$\{19,23,4,9,10,12\}$和$\{13,17,29,3,4,6\}$两个区组之外的所有区组中,有六个属性中的两个。

Marshall Hall,Jr的专著*Combinatorial Theory*(Waltham,Mass,Blaisdell,1967)详细地论述了区组设计以及有关模式的理论。尽管这些组合配置是非常漂亮的,但是迄今对于信息检索的主要应用,还只是减少当使用复合的反表时引起的冗余性;而且David.K.Chow[*Information and Control* **15**(1969),377～396]已经发现,甚至不使用组合设计也可以得到同样的减少效果。

简史和文献目录 关于辅助键码检索技术的头一篇公开的论文,是由L.R.Johnson在*CACM* **4**(1961),218～222上发表的。多重表系统是由Noah.S.Prywes,H.J.Gray,W.I.Landauer, D.Lefkowitz和S.Litwin大约在同一时期独立地提出的;参见*IEEE Trans. on Communication and Electronics* **82**(1963),488～492。另一篇相当早的并对后来的工作有影响的著作是D.R.Davis和A.D.Lin的,*CACM* **8**(1965),243～246。

自那以后，关于这一课题的大量文献迅速增长，但大多数都涉及用户接口和程序设计语言的考虑，这些不属于本书的范围。除了已经引证的论文外，下列已发表的论文是当作者于 1972 年撰写这一节时发现最有用的：Jack Minker 和 Jerome Sable,*Ann. Rev. of Information Science and Technology* **2**(1967),123～160,Robert E. Bleier *Proc. ACM Nat. Conf.* **22** (1967),41～49;Jerome A. Feldman 和 Paul D. Rovner,*CACM* **12** (1969),439～449;Burton H. Bloom, *Proc. ACM Nat. Conf.* **24** (1969),83～95;H. S. Heaps 和 L. H. Thiel,*Information Storage and Retrieval* **6** (1970),137～153;林耀森和林辉,*Proc. ACM Nat. Conf.* **26** (1971),349～356。关于人工卡片文件系统的一个很好的综述，见 C. P. Bourne 的*Methods of Information Handling*（纽约：Wiley,1963），第 5 章。平衡的文件系统最初是由 C. T. Abraham, S. P. Ghosh 以及 D. K. Ray-Chaudhuri 于 1965 年提出的；参见 R. C. Bose 和 Gary. G. Koch 的论文,*SIAM. J. Appl. Math.* **17** (1969),1203～1214。

以上已经讨论了关于多属性数据的大多数经典的算法，它们是众所周知的在实践中很重要的算法。但在本书的下一版本中，打算还讨论另外一些课题，这些课题包括：

• E. M McCreight 介绍了优先查找树[*SICOMP* **14** (1985),257～276]，它们被专门设计来表示区间的动态变化类的交，并且来处理形如“求满足 $x_0 \leqslant x \leqslant x_1$ 和 $y \leqslant y_1$ 的所有记录”的范围查询。（注意 y 的下限必须是 $-\infty$，但 x 在两边都可以是有限的。）

• M. L. Fredman 已经证明了若干基本的下限，它们表明，无论是用什么数据结构，在最坏情况下，N 个相互混合的插入、删除和 k 维范围查询的一个系列必须花费 $\Omega(N(\log N)^k)$ 个操作。参见*JACM* **28** (1981),696～705;*SICOMP* **10** (1981),1～10;*J. Algorithms* **1** (1981),77～87。

有关在文本串中模式匹配和近似模式匹配的基本算法将在第 9 章中讨论。

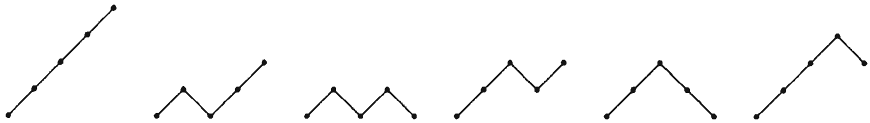
指出这样一点是有趣的，即人脑在辅助键码检索方向要比计算机好得多；事实上，人们发现从只有片断的信息来认记面孔和曲调比较容易，而计算机则几乎全然不可能做这件事。因此，将来某一天会发现全新的机器设计方法，它能一劳永逸地解决辅助键码检索的问题，因而使本节完全过时，这并不是不可能的。

习　题

▶1.[*M27*] 设 $0 \leqslant k \leqslant n/2$。证明下列构造产生 $\binom{n}{k}$ 个 $\{1,2,\cdots,n\}$ 的排列，使得对于 $t \leqslant k$ 或 $t \geqslant n-k$，$\{1,2,\cdots,n\}$ 的每个 t 元子集至少作为一个排列的前 t 个元素出现：考虑平面中从 $(0,0)$ 到 (n,r) 的一条通路，这里 $r \geqslant n-2k$，在这条通路中第 i 步是从 $(i-1,j)$ 到 $(i,j+1)$ 或者到 $(i,j-1)$；仅当 $j \geqslant 1$ 时才允许后一种可能性，因此这条通路决不会跑到 x 轴下边。恰有 $\binom{n}{k}$ 个这样

的通路。对于每条这种类型的通路,利用开始时为空的三个表列构造一个排列如下:对于 $i=1,2,\cdots,n$,如果这条通路的第 i 步往上去,就把号 i 放到表 B 中;如果这步往下,就把 i 放到表 A 中,而且把当前表 B 中最大的元素放到表 C 中。得到的排列是表 A,然后表 B,然后表 C 的最后内容,每个表处于递增的次序。

例如,当 $n=4$ 和 $k=2$ 时,由这个过程确定的六条通路和排列为



|1 2 3 4|　　2|3 4|1　　2 4||1 3　　3|1 4|2　　3 4||1 2　　4|1 2|3

(垂直线示出表 A、B 和 C 之间的分界线。这六个排列对应于(8)中的复合属性)。

提示:通过从(0,0)到$(n,n-2t)$的一条通路表示每 t 个元素的子集 S,如果 $i\notin S$,它的第 i 个步骤从$(i-1,j)$进行到$(i,j+1)$,如果 $i\in S$,则进行到$(i,j-1)$。把每条这样的通路转换成有上述特殊形式的一条适当通路。

2.[*M25*](Sakti.P.Ghosh) 试求用于访问记录的一个表 $r_1r_2\cdots r_l$ 的最小长度 l,使得对三个二进值的辅助键码进行的任意包含查询 * * 1, * 1 * ,1 * * , * 11,1 * 1,11 * ,111 的所有回答的集合,都将出现于连续的单元 $r_i\cdots r_j$ 中。

3.[*19*] 在表 2 中,什么包含查询将引起(a)旧式家常糖饼,(b)燕麦枣椰条饼在假情报中出现?

4.[*M30*] 假设每个记录有 r 个不同的属性,它们是从 n 位字段的 $\binom{n}{k}$ 个 k 位二进位代码中随机选择的,且查询包括 q 个不同的但除此以外是随机的属性,试求(11)中概率的精确公式(如果这些公式没有简化,也不要吃惊)。

5.[*40*] 当对子串的查找使用 Harrison 技术时,实验各种方法来避免正文的冗余性。

▶6.[*M20*] 指明 t 个二进位的 m 位基本查询的总数是 $s=\binom{m}{t}2^t$。如果像在(13)中那样的一个组合散列函数把这些查询分别地转化成为 $l_1,l_2,\cdots,l_s$ 个位置,则 $L(t)=(l_1+l_2+\cdots+l_s)/s$ 是每个查询的平均位置数。[例如,在(13)中我们有 $L(3)=1.75$。]

现在考虑对一个(m_1+m_2)个二进位字段的合成散列函数,它是由用一个散列函数映射前 m_1 位,用另一个散列函数映射剩下的 m_2 位形成的,其中 $L_1(k)$和 $L_2(k)$是每个查询对应的平均位置数。试借助于 L_1 和 L_2,求表达的合成函数 $L(t)$的一个公式。

7.[*M24*](R.L.Rivest) 如同上题中定义的那样,对于下列组合的散列函数,求函数 $L(t)$:

(a) $m=3,n=2$	(b) $m=4,n=2$
0 0 * → 0	0 0 * * → 0
1 * 0 → 1	* 1 * 0 → 1
* 1 1 → 2	* 1 1 1 → 2
1 0 1 → 3	1 0 1 * → 2
0 1 0 → 3	* 1 0 1 → 3
	1 0 0 * → 3

8.[*M32*](R.L.Rivest) 考虑类似于(10)中的所有 $2^t\binom{m}{t}$ 个基本的 m 个二进位查询的集合 $Q_{t,m}$,其中恰有 t 个指定的二进位。给定 m 个二进位的记录的一个集合 S,令 $f_t(S)$表示在 $Q_{t,m}$

中查询的个数，它们的回答包含 S 的一个成员；令 $f_t(s,m)$ 是对于 $0\leqslant s\leqslant 2^m$，对于所有有 s 个元素的所有这样的集合 S 的极小值 $f_t(S)$。约定 $f_t(0,0)=0$ 和 $f_t(1,0)=\delta_{t,0}$。

a)证明，对于所有 $t\geqslant 1,m\geqslant 1$ 以及对于 $0\leqslant s\leqslant 2^m$，$f_t(s,m)=f_t(\lceil s/2\rceil,m-1)+f_{t-1}(\lceil s/2\rceil,m-1)+f_{t-1}(\lfloor s/2\rfloor,m-1)$

b)考虑从 2^m 个可能的记录到 2^n 个表列的任何组合散列函数 h，且每个表列对应于 2^{m-n} 个记录。如果 $Q_{t,m}$ 中的每个查询都是同等可能的，则对于每个查询需要考察的表列的平均数是 $1/2^t\binom{m}{t}$ 乘以

$$\sum_{Q\in Q_{t,m}}(\text{对于 }Q\text{ 考察的表列})=\sum_{\text{表列}S}(\text{同 }S\text{ 有关的 }Q_{t,m}\text{ 的查询})\geqslant 2^n f_t(2^{m-n},m)$$

证明，当每个表列是一个“子立方体”时，在这个下限可达到的意义下，h 是最优的；换句话说，当每个表列对应于满足带有恰好 n 个指定的二进位的某个基本查询的一个记录的情况下，等式成立。

9.[*M20*] 证明当 $v=3^n$ 时，所有形如

$$\{(a_1\cdots a_{k-1}0b_1\cdots b_{n-k})_3,(a_1\cdots a_{k-1}1c_1\cdots c_{n-k})_3,(a_1\cdots a_{k-1}2d_1\cdots d_{n-k})_3\},1\leqslant k\leqslant n$$

的三元组的集合形成一个 Steiner 三元组系统，其中诸 a，诸 b，诸 c 和诸 d 取遍 0，1 和 2 的所有使得 $b_j+c_j+d_j\equiv 0$(modulo 3)的组合，其中 $1\leqslant j\leqslant n-k$。

10.[*M32*] (Thomas. P. Kirkman, *Cambridge and Dublin Math. Journal* **2**(1847), 191～204) 我们说阶为 v 的一个 *Kirkman* 三元组系统是把 $v+1$ 个对象 $\{x_0,x_1,\cdots,x_v\}$ 变成为三元组的一个安排，使得对于 $i\neq j$，每个对偶 $\{x_i,x_j\}$ 恰出现于一个三元组中，但是对于 $0\leqslant i<v$，v 个对偶 $\{x_i,x_{(i+1)\bmod v}\}$ 决不出现于同一个三元组中。例如

$$\{x_0,x_2,x_4\},\{x_1,x_3,x_4\}$$

是阶为 4 的一个 Kirkman 三元组系统。

a)证明仅当 $v \bmod 6=0$ 或 4 时一个 Kirkman 三元组系统才能存在。

b)给定 v 个对象 $\{x_1,\cdots,x_v\}$ 上的一个 Steiner 三元组系统 S，证明下列构造产生 $2v+1$ 个对象上的另一个 Steiner 系统 S' 以及阶为 $2v-2$ 的一个 Kirkman 三元组系统 K'：S' 的三元组是 S 的三元组加上

i) $\{x_i,y_j,y_k\}$，其中 $j+k\equiv i$(modulo v) 及 $j<k,1\leqslant i,j,k\leqslant v$；

ii) $\{x_i,y_j,z\}$，其中 $2j\equiv i$(modulo v)，$1\leqslant i,j\leqslant v$。

K' 的三元组是 S' 的那些三元组减去含 y_1 和/或 y_v 的那些三元组。

c)给定 $\{x_0,x_1,\cdots,x_v\}$ 上的一个 Kirkman 三元组系统，其中 $v=2u$，证明下列构造产生 $2v+1$ 个对象的一个 Steiner 三元组系统 S'，以及阶为 $2v-2$ 的一个 Kirkman 三元组系统 K'：S' 的三元组是 K 的三元组加上

i) $\{x_i,x_{(i+1)\bmod v},y_{i+1}\}$，$0\leqslant i<v$；

ii) $\{x_i,y_j,y_k\}$，$j+k\equiv 2i+1$(modulo $v-1$)，$1\leqslant j<k-1\leqslant v-2$，$1\leqslant i\leqslant v-2$；

iii) $\{x_i,y_j,y_v\}$，$2j\equiv 2i+1$(modulo $v-1$)，$1\leqslant j\leqslant v-1$，　$1\leqslant i\leqslant v-2$；

iv) $\{x_0,y_{2j},y_{2j+1}\}$，$\{x_{v-1},y_{2j-1},y_{2j}\}$，$\{x_v,y_j,y_{v-j}\}$，对于 $1\leqslant j<u$；

v) $\{x_v,y_u,y_v\}$。

K' 的三元组是 S' 的三元组减去包含 y_1 和/或 y_{v-1} 的那些三元组。

d)利用上面的结果证明阶为 v 的 Kirkman 三元组系统对于形如 $6k$ 或 $6k+4$ 的所有 $v\geqslant 0$ 存

在，且 v 个对象的 Steiner 三元组系统对于形如 $6k+1$ 或 $6k+3$ 的所有 $v\geqslant 1$ 存在。

11.［*M25*］正文描述了同包含查询相联系时 Steiner 三元组系统的用法。为了把这推广到所有基本的查询中去，定义下列概念是自然的：阶为 v 的一个互补三元组系统是把 $2v$ 个对象 $\{x_1,\cdots,x_v,\bar{x}_1,\cdots,\bar{x}_v\}$ 编成下面这样一些三元组的一个安排，即每对对象恰在一个三元组中一起出现，但互补的对 $\{x_i,\bar{x}_i\}$ 决不一起出现。例如

$$\{x_1,x_2,x_3\},\{x_1,\bar{x}_2,\bar{x}_3\},\{\bar{x}_1,x_2,\bar{x}_3\},\{\bar{x}_1,\bar{x}_2,x_3\}$$

是阶为 3 的一个互补三元组系统。

证明对于所有不是形如 $3k+2$ 的 $v\geqslant 0$，都存在阶为 v 的互补三元组系统。

12.［*M23*］继续习题 11，构造阶为 7 的一个互补四元组系统。

13.［*M25*］构造类似于习题 9 的三元组系统的具有 $v=4^n$ 个元素的四元组系统。

14.［*28*］试讨论从四叉树、$k-\mathrm{d}$ 树以及类似于图 45 的邮局树删去节点的问题。

15.［*HM30*］P. Elias 给定 m 个二进位记录的一个很大集合，假设我们要来找一个记录，在大多数二进位一致的意义下，它最接近于一个给定的查找变元。假定给定了 2^n 个元素的一个 m 个二进位的 t-误差校正码，而且每个记录已经被"散列到"对应于最接近的码字的 2^n 个表之一，试设计用于有效地解决这个问题的一个算法。

▶16.［*25*］（W. H. Kautz 和 R. C. Singleton）证明阶为 v 的一个 Steiner 三元组系统可用以构造每个有 v 个二进位的 $v(v-1)/6$ 个代码字，使得任一代码字不被包含在其它两个代码字的叠加码中。

▶17.［*M30*］试考虑下列把 $(2n+1)$ 个二进位的键码 $a_{-n},\cdots,a_0,\cdots,a_n$ 减少成 $(n+1)$ 个二进位的桶地址 $b_0,\cdots,b_n$ 的方法

$b_0\leftarrow a_0$；

如果 $b_{k-1}=0$ 则 $b_k\leftarrow a_{-k}$ 否则 $b_k\leftarrow a_k$，对于 $1\leqslant k\leqslant n$。

a）试描述出现在桶 $b_0\cdots b_n$ 中的诸键码；

b）在有指定的 t 个二进位的一个基本查询中，需要考察的桶的最大个数是多少？

▶18.［*M35*］（相关区组设计）一个类似(13)的 m 元组的集合，且在 2^n 个行的每行中恰有 $m-n$ 个 *，如果每个列包含相同个数的 *，而且如果在每对的行中都有在某列中的一个"不匹配"（0 对 1），则称这个集合为 ABD(m,n)。于是每个 m 个二进位的数都将恰好匹配一行。例如，(13)是一个 ABD(4,3)。

a）试证明，除非 m 是 $2^{n-1}n$ 的一个因子而且 $n^2\geqslant 2m(1-2^{-n})$，否则，ABD$(m,n)$是不可能的。

b）如果 ABD 的一个行包含有奇数个 1，则说它是奇校验的。试证明，对于一个 ABD(m,n)中的 $m-n$ 个列的每一个选择，在这些列中具有 * 的奇校验的行的个数等于偶校验的行的个数，特别是，星号的每种模式必然在偶数个行中出现。

c）通过排列和/或对列取补，求出由(13)不能得到的一个 ABD(4,3)。

d）构造一个 ABD(16,9)。

e）构造一个 ABD(16,10)，由部分 d）的 ABD(16,9)开始，而不是由(15)的 ABD(8,5)开始。

19.［*M22*］如同在(14)中已经分析了(13)那样，试分析(15)的 ABD(8,5)；对于有 k 个二进位未指定的一个普通的查找来说，必须查找 32 个位置中的多少个位置？在最坏情况下，必须查找多少个位置？

20.［*M47*］当 $n=5$ 或 $n=6$ 时，求所有 ABD(m,n)。

本书的下一版本的新的6.6节打算专门讨论“持久的数据结构”。持久的数据结构能够以这样一种方式来表示变动的信息，即过去的历史可以有效地重新构造出来。换句话说，我们可以做许多的插入和删除，但我们仍旧能进行查找，就好像在一个给定的时间之后未曾做过修改似的。对于这一课题有关的参考文献包括下列的一些论文：

- J.K. Mullin, *Comp. J.* **24** (1981), 367～373;
- M.H. Overmars, *Lecture Notes in Comp. Sci*, **156** (1983), 第9章;
- E.W. Myers, *ACM Symp. Principles of Prog. Lang.*, **11** (1984), 66～75;
- B. Chazelle, *Information and Control*, **63** (1985), 77～99;
- D. Dobkin 和 J.I. Munro, *J. Algorithms*, **6** (1985), 455～465;
- R. Cole, *J. Algorithms*, **7** (1986), 202～220;
- D. Field, *Information Processing Letters*, **24** (1987), 95～96;
- C.W. Fraser 和 E.W. Myers, *ACM Trans. Prog. Lang. and Systems* **9** (1987), 277～295;
- J.R. Driscoll, N. Sarnak, D.D. Sleator 和 R.E. Tarjan, *J. Comp. Syst. Sci.*, **38** (1989), 86～124;
- R.B. Dannenberg, *Software Practice & Experience*, **20** (1990), 109～132;
- J.R. Driscoll, D.D.K. Sleator 和 R.E. Tarjan, *JACM* **41** (1994), 943～959。

Instruction tables [programs] will have to be made up
by mathematicians with computing experience
and perhaps a certain puzzle soiving ability.
There will probably be a great deal of work of this kind to be done,
for every known process has got to be
translated into instruction table form at some stage. ...
This process of constructing instruction tables should be very fascinating.
There need be no real danger of it ever becoming a drudge,
for any processes that are quite mechanical
may be turned over to the machine itself.

指令表[程序]将要由具有计算经验以及或许还有某种解决智力游戏问题的能力的数学家来编制。
大概将有大量的这一类型的工作需要完成，因为在每个阶段，每个已知的过程都需要被翻译成指令表的形式。
构造指令表的这一过程将非常迷人。它不含有变成单调乏味的实际危险，因为任何那些十分机械化的过程都可以交给机器本身来完成。

——ALAN M. TURING(1945)

习 题 答 案

"I have answered three questions, and that is enough"
Said his father, "don't give yourself airs!
Do you think I can listen all day to such stuff?
Be off, or I'll kick you down stairs!"
"我已经回答了三个问题，而这就够了，"他的父亲说，"不要装腔作势！
你是不是以为我能整天地听这样的废话？滚开，
否则我将一脚把你踢下楼去！"
——LEWIS CARROLL, *Alice's Adventures Under Ground* (1864)

关于习题的说明

1. 对于一个有数学修养的读者来说是一个普通的问题。

3. 见 W. J. LeVeque, *Topics in Number Theory* **2**(Reading, Mass.: Addison-Wesley, 1956), 第 3 章。P. Ribenboim, *13 Lectures on Fermat's Last Theorem* (New York: Springer-Verlag, 1979), A. Wiles, *Annals of Mathematics* **141**(1995), 443～551。

第 5 章

1. 设 $p(1),\cdots,p(N)$ 和 $q(1),\cdots,q(N)$ 是满足这些条件的不同的排列，且设 i 是使 $p(i)\neq q(i)$ 的极小值。则对某个 $j>i$ 有 $p(i)=q(j)$ 和对于某个 $k>i$，有 $q(i)=p(k)$。由于 $K_{p(i)}\leqslant K_{p(k)}=K_{q(i)}\leqslant K_{q(j)}=K_{p(i)}$，我们有 $K_{p(i)}=K_{q(i)}$；因此由稳定性 $p(i)<p(k)=q(i)<q(j)=p(i)$，矛盾。

2. 是的，如果排序操作都是稳定的话。(如果它们不稳定，则我们不能这么说。) Alice 和 Chris 肯定有相同的结果；而且 Bill 也是，因为稳定性表明，在他的结果中当主键码相等时，次键码是按非减次序排列的。

形式上，假定 Bill 对次键码进行排序之后，得到结果 $R_{p(1)}\cdots R_{p(N)}=R'_1\cdots R'_N$，然后在对主键码进行排序之后得到 $R'_{q(1)}\cdots R'_{q(N)}=R_{p(q(1))}\cdots R_{p(q(N))}$；我们要证明：对于 $1\leqslant i<N$，$(K_{p(q(i))},k_{p(q(i))})\leqslant(K_{p(q(i+1))},k_{p(q(i+1))})$。如果 $K_{p(q(i))}\neq K_{p(q(i+1))}$，则我们有 $K_{p(q(i))}<K_{p(q(i+1))}$；又如果 $K_{p(q(i))}=K_{p(q(i+1))}$，则 $K'_{q(i)}=K'_{q(i+1)}$，因此 $q(i)<q(i+1)$，因此 $k'_{q(i)}\leqslant k'_{q(i+1)}$，即 $k_{p(q(i))}\leqslant k_{p(q(i+1))}$。

3. 我们总能把具有相等键码的记录送到一起，并保持它们的相对次序，在进一步的操作中把这些记录组当作一个单位来处理；因此，我们可以假定，所有的键码都

是不同的。设 $a<b<c<a$;则我们可以把这些记录安排成头三个键为 abc,bca,或 cab。现在如果 $N-1$ 个不同的键码可以以三种方法加以排序,则 N 个也能;因为如果 $K_1<\cdots<K_{N-1}>K_N$,则对于某个 i 总有 $K_{i-1}<K_N<K_i$,或者 $K_N<K_1$。

4. 首先比较没有大小写区别的字,然后使用大小写来打破僵局。更确切地说,以对偶(α',α)来代替每个字 α,其中 α'是由 α 通过映射 A→a,…,Z→z 得到的;然后按字典次序对这些对偶进行排序。例如,这个过程给出 tex< Tex < TeX < TEX< text。

字典也必须处理加重音的字母,前缀,后缀以及略写;例如

a < A < Å < a-< a. <-a < A-< A. < aa < a.a.

< āa < āā < AA < A.A. < AAA < … < zz < Zz < ZZ < zzz < ZZZ

在这个更一般的情况下,我们通过映射 ā→a,Å→a 等,以及去掉破折号及句点等得到 α'。

5. 设 $\rho(0)=0$ 和 $\rho((1\alpha)_2)=1\rho(|\alpha|)\alpha$;这里$(1\alpha)_2$ 是一个正整数的通常的二进表示,而$|\alpha|$是串 α 的长度。我们有 $\rho(1)=10$,$\rho(2)=1100$,$\rho(3)=1101$,$\rho(4)=1110000$,…,$\rho(1009)=11110100111110001$,…,$\rho(65536)=1^50^{24}$,…,$\rho(2^{65536})=1^60^{65560}$,等。$\rho(n)$的长度是

$$|\rho(n)| = \lambda(n)+\lambda(\lambda(n))+\lambda(\lambda(\lambda(n)))+\cdots+\lg^* n+1$$

其中 $\lambda(0)=0$,对于 $n\geqslant 1$,$\lambda(n)=\lfloor \lg n\rfloor$。而且 $\lg^* n$ 是使得 $\lambda^{[m]}(n)=0$ 的最小整数 $m\geqslant 0$。[这个构造是由 V.I.Levenshtein,*Problemy Kibernetiki* ,**20**(1968),173~179 给出的。也请参见由 D.A.Klarner 主编*The Mathematical Gardner*(加州 Belmont:Wadsworth International,1981) 310~325,D.E.Knuth 的论文。]

6. 溢出是可能的,而且它能导致一个假的相等指示。他应该编写,LDA A;CMPA B 并检查比较指示器(不可能通过减法来做全字长的比较,这实质上是许多计算机上的一个问题;这是在 MIX 的指令系统中包括 CMPA,…,CMPX 的主要原因)。

7.

```
COMPARE STJ    9F
1H      LDX    A,1
        CMPX   B,1
        JNE    9F
        DEC1   1
        J1P    1B
9H      JMP    *
```
▌

8. 解法 1,以恒等式 $\min(a,b)=\frac{1}{2}(a+b-|a-b|)$为基础:

```
SOL1 LDA  A            STA  A1      a = 2a1 + a2
     SRAX 5            STX  A2      |a2| ≤ 1
     DIV  =2=          LDA  B
```

```
SRAX  5                         SLAX  5
DIV   =2=                       MUL   AB2
STA   B1    b=2b1+b2            STX   AB3    (a2-b2) sign(a-b)
STX   B2    |b2|≤1                    LDA    A2
LDA   A1                        ADD   B2
SUB   B1    不可能溢出          SUB   AB3
STA   AB1   a1-b1               SRAX  5
LDA   A2                        DIV   =2=
SUB   B2                        ADD   A1
STA   AB2   a2-b2               ADD   B1     不可能溢出
SRAX  1                         SUB   AB1(1:5)
ADD   AB1                       STA   C       ▌
ENTX  1
```

解法 2,以变址能以一种巧妙的方式引起交换这一事实为基础:

```
SOL2  LDA   A
      STA   C
      STA   TA
      LDA   B
      STA   TB
```

现在重复下列代码 k 次,这里 $2^k > 10^{10}$:

```
LDA   TA        LDA   TB        INC1  0,2
SRAX  5         SRAX  5         INC1  0,2
DIV   = 2 =     DIV   = 2 =     INC1  0,2
STX   TEMP      STX   TEMP      LD3   TMIN,1
LD1   TEMP      LD2   TEMP      LDA   0,3
STA   TA        STA   TB        STA   C
```

(这从右到左地扫描 a 和 b 的二进表示,并保持它们的符号。)最后,程序以下列一个表结束

```
      HLT
      CON   C   -1   -1
      CON   B    0   -1
      CON   B   +1   -1
      CON   A   -1    0
TMIN  CON   C    0    0
      CON   B    1    0
      CON   A   -1    1
      CON   A    0    1
      CON   C    1    1   ▌
```

9. 由容斥原理*(习题 1.3.3-26)得到 $\sum_j \binom{r+j-1}{j}(-1)^j\binom{N}{r+j}x^{r+j}$。这是一个 β 分布,也可写成为 $r\binom{N}{r}\int_0^x t^{r-1}(1-t)^{N-r}\mathrm{d}t$。

10. 将磁带的内容进行排序,然后进行计数(当排序进行时,某些排序方法可用来方便地筛除其键码出现一次以上的记录)。

11. 给每一个人指定一个标识号,这个号必须在所有涉及他的表格中出现。以这个标识号为键码,分别对信息表格和赋税的表格排序。以 $R_1,\cdots,R_N$ 表示排好序的赋税表格,其键码为 $K_1<\cdots<K_N$。(对于相等的键码不应该有两张赋税表格。)增加一个新的其键码为∞的记录,这是第$(N+1)$个记录并且置 $i\leftarrow1$。然后对于信息文件中的每个记录,校验它是否已经报告如下:设 K 表示正在处理的信息表上的键码。

a)如果 $K>K_i$,则 i 增加 1,并且重复这个步骤。

b)如 $K<K_i$ 或者如果 $K=K_i$ 而且信息不反映到赋税表格 R_i 上,则标出一个错误。尝试来做这全部工作而不浪费纳税人的钱。

12. 一个方法是附加键码(j,i)到条目 $a_{i,j}$上,而且利用字典编辑顺序来排序,然后删去键码。(当能够给出重新编序的一个简单公式时,一个类似的思想可以用来得到对于信息的想要的任何重新排序。)

在这个问题所考虑的特殊情况下,"平衡的两路合并排序"方法以这样一种简单的方式来处理键码,即不必明显地在磁带上写出任何键码来。给定一个 $n\times n$ 矩阵,我们可以这样做:首先在磁带 1 上放置奇数编号的行,在磁带 2 上放置偶数编号的行,等等,得到:

磁带 1:$a_{11}a_{12}\cdots a_{1n}a_{31}a_{32}\cdots a_{3n}a_{51}a_{52}\cdots a_{5n}\cdots$

磁带 2:$a_{21}a_{22}\cdots a_{2n}a_{41}a_{42}\cdots a_{4n}a_{61}a_{62}\cdots a_{6n}\cdots$

然后反绕这些磁带,而且同步地处理它们,以得到

磁带 3:$a_{11}a_{21}a_{12}a_{22}\cdots a_{1n}a_{2n}a_{51}a_{61}a_{52}a_{62}\cdots a_{5n}a_{6n}\cdots$

磁带 4:$a_{31}a_{41}a_{32}a_{42}\cdots a_{3n}a_{4n}a_{71}a_{81}a_{72}a_{82}\cdots a_{7n}a_{8n}\cdots$

反绕这些磁带,并且同步地处理它们,以得到

磁带 1:$a_{11}a_{21}a_{31}a_{41}a_{12}\cdots a_{42}\cdots a_{4n}a_{9,1}\cdots$

磁带 2:$a_{51}a_{61}a_{71}a_{81}a_{52}\cdots a_{82}\cdots a_{8n}a_{13,1}\cdots$

等等,直到$\lceil\lg n\rceil+1$ 遍扫描之后得到所希望的转置。

13. 一个方法是附加随机的不同的键码值,按这些键码进行排序,然后抛弃这些键码。(参考习题 12;在 3.4.2 小节中讨论了为得到一个随机的样品的一个类似方法。)另外一项技术需要差不多同样多的工作量但显然不太强求随机数生成程序

* 容斥原理又译为包括和排除原理。——译注

的精确度，它附加一个在范围 $0 \leqslant K_i < N - i$ 内的随机整数到 R_i 上，然后利用习题 5.1.1-5 的技术重新安排。

14. 通过一个字符转换表，你能设计一个按字典顺序的比较程序，它模拟在其它机器上所用的次序。或者，你可以建立人工的键码，它不同于实际的字符但是给出所希望的次序。后一种方法有这样一个优点：它仅仅需要做一次。但是它花费较多的空间而且要求对整个键码的转换。前一种方法通常可以仅仅通过转换键码字的一个或两个字母，就确定了比较的结果；在排序的后边诸阶段，这个比较将仅仅在几乎相等的键码之间进行。因此在前一种方法中，在转换字母之前校验字母的相等性也许是有优点的。

15. 对于这个问题，仅需扫视这个文件一次并保持 50 个左右个别的计数。但如用“城市”来代替“州”，而且如果城市的总数十分大，则对于城市的名称进行排序将是一个好的想法。

16. 像在习题 15 中一样它依赖于问题的大小。如果高速的存储器能放得下交叉引用的全部记录，则最好的方法大概是使用一个符号表算法（第六章），而且每个标识符与一个引用表的表头相联系。对于更大的问题，建立一个记录文件，为放置到索引中去的每个交叉引用条目都设置一个记录，并且对此文件进行排序。

17. 每张卡片上设一个“阴形键码”，若以通常简单的方式按字典顺序对键码进行排序，就能定义所要的次序。这个键码应由图书馆职员提供，而且当它首次进入这个系统时就附加到卡片数据上，但对于通常用户来说它是看不见的。则一个可能的键码使用以下的双字母代码把字彼此分开：

␣0	键码的结束
␣1	交叉引用的结束
␣2	姓的结束
␣3	多个姓的连字号
␣4	作者名的结束
␣5	地名的结束
␣6	标题的结束
␣7	书的题目的结束
␣8	词之间的空格

对于给定的例子，我们将得出如下结果（仅仅给出头 25 个字符）：

ACCADEMIA␣8NAZIONALE␣8DEI
ACHTZEHNHUNDERT ZWOLF␣8EIN
BIBLIOTHEQUE␣8D␣8HISTOIRE
BIBLIOTHEQUE␣8DES␣8CURIOS
BROWN␣2J␣8CROSBY␣4␣0
BROWN␣2JOHN␣4␣0
BROWN␣2JOHN␣4MATHEMATICIA
BROWN␣2JOHN␣4OF␣8BOSTON␣0
BROWN␣2JOHN␣41715␣0
BROWN␣2JOHN␣41715␣6␣0
BROWN␣2JOHN␣41761␣0
BROWN␣2JOHN␣41810␣0

BROWN␣3WILLIAMS␣2REGINALD
BROWN␣8AMERICA␣7␣0
BROWN␣8AND␣8DALLISONS␣8NE
BROWNJOHN␣2ALAN␣4␣0
DEN␣2VLADIMIR␣8EDUARDOVIC
DEN␣7␣0
DEN␣8LIEBEN␣8LANGEN␣8TAG␣
DIX␣2MORGAN␣41827␣0
DIX␣8HUIT␣8CENT␣8DOUZE␣8O
DIX␣8NEUVIEME␣8SIECLE␣8FR
EIGHTEEN␣8FORTY␣8SEVEN␣8I
EIGHTEEN␣8TWELVE␣8OVERTUR
I␣8AM␣8A␣8MATHEMATICIAN␣7
I␣8B␣8M␣8JOURNAL␣8OF␣8RES
I␣8HA␣8EHAD␣7␣0
IA␣8A␣8LOVE␣8STORY␣7␣0
INTERNATIONAL␣8BUSINESS␣8
KHUWARIZMI␣2MUHAMMAD␣8IBN
LABOR␣7A␣8MAGAZINE␣8FOR␣8
LABOR␣8RESEARCH␣8ASSOCIAT
LABOUR␣1␣0
MACCALLS␣8COOKBOOK␣7␣0
MACCARTHY␣2JOHN␣41927␣0
MACHINE␣8INDEPENDENT␣8COM
MACMAHON␣2PERCY␣8ALEXANDE
MISTRESS␣8DALLOWAY␣7␣0
MISTRESS␣8OF␣8MISTRESSES␣
ROYAL␣8SOCIETY␣8OF␣8LONDO
SAINT␣8PETERSBURGER␣8ZEIT
SAINT␣8SAENS␣2CAMILLE␣418
SAINTE␣8MARIE␣2GASTON␣8P␣
SEMINUMERICAL␣8ALGORITHMS
UNCLE␣8TOMS␣8CABIN␣7␣0
UNITED␣8STATES␣8BUREAU␣8O
VANDERMONDE␣2ALEXANDER␣8T
VANVALKENBURG␣2MAC␣8ELWYN
VONNEUMANN␣2JOHN␣41903␣0
WHOLE␣8ART␣8OF␣8LEGERDEMA
WHOS␣8AFRAID␣8OF␣8VIRGINI
WIJNGAARDEN␣2ADRIAAN␣8VAN

这个辅助的键码应该接有卡片数据，以便正确地区别具有相同辅助键码(例如 Sir John = John)的不相等卡片。注意“Saint-Saëns”是一个连接的名字而不是一个复合名。al-Khuwārizmī 的出生年份应当以比如说带一个前导 0 的␣40779 给出。(这个方案在 9999 年之前都将有效，而在该年之后世界将面临巨大的软件危机。)

对于这个例子的仔细研究揭示如何处理在人机交互中需要的其它非寻常类型的次序。

18．例如，对于 $u\leqslant v\leqslant w$，$x\leqslant y\leqslant z$ 我们可以造包含$(u^6+v^6+w^6) \bmod m$ 和 $(z^6-x^6-y^6) \bmod m$ 的值的两个文件，这里 m 是计算机的字长。对文件进行排序并且把重复的挑出来，然后对重复者施以进一步的检验。(某些对小素数求模的同余式也可用来对 u,v,w,x,y,z 设置进一步的限制。)

19．一般地说，有如下方法找出满足 $x_i+x_j=c$ 的所有数对 $\{x_i,x_j\}$，这里 c 是给定的：对文件进行排序，使得 $x_1<x_2<\cdots<x_N$。置 $i\leftarrow1$，$j\leftarrow N$，然后重复下列操作直到 $j\leqslant i$：

如果 $x_i+x_j=c$，则输出 $\{x_i,x_j\}$，置 $i\leftarrow i+1$，$j\leftarrow j-1$；

如果 $x_i + x_j < c$，则置 $i \leftarrow i + 1$；

如果 $x_i + x_j > c$，则置 $j \leftarrow j - 1$。

最后，如果 $j = i$ 而且 $2x_i = c$，则输出 $\{x_i, x_j\}$。这个过程就像习题 18 的方法那样：我们实质上造两个有序文件，一个包含 $x_1, \cdots, x_N$，另一个包含 $c - x_N, \cdots, c - x_1$，而且校验重复。但第二个文件在这种情况下不需要明显地形成。当 c 为奇数时，另一个方法是对一个诸如(x 奇$\Rightarrow x$，x 偶$\Rightarrow c - x$)的键码进行排序。

当给定 t 和两个排好序的文件 $x_1 \leqslant \cdots \leqslant x_m$，$y_1 \leqslant \cdots \leqslant y_n$ 时，一个类似的算法可用来求 $\max\{x_i + x_j \mid x_i + x_j \leqslant c\}$；或者比如说，来求 $\min\{x_i + y_j \mid x_i + y_j > t\}$。

20. 某些方案是：(a)对于满足 $1 \leqslant i < j \leqslant 1000$ 的 499 500 个对偶 i, j 的每一个，置 $y_1 \leftarrow x_i \oplus x_j$，$y_2 \leftarrow y_1 \wedge (y_1 - 1)$，$y_3 \leftarrow y_2 \wedge (y_2 - 1)$；然后，当且仅当 $y_3 = 0$ 时，打印(x_i, x_j)。这里$\oplus$表示“异或”，$\wedge$ 表示“按位与”。(b)由每个原来的字 x_i 形成 31 个项，其中包括 x_i 本身及在一个位置上不同于x_i 的另外 30 个字，这样得到一个共有 31 000 个项的文件。对这个文件进行排序，而且考察重复性。(c)对于

i)它们的头 10 个二进位一致的所有字对；

ii)它们中间 10 个二进位(但不是头 10 个二进位)中一致的所有字对；

iii)它们后 10 个二进位(但既不是头 10 个二进位也不是中间 10 个二进位)一致的所有字对。

作类似于(a)的测试。这包含三个数据排序，并且每次使用一个确定的 10 个二进位的键码。如果原来的字是随机分布的，则在三种情况的每一种中预期的对偶数目至多是 $499500/2^{10}$，它小于 500。

21. 首先准备好包含所有五个字母的英文单词的一个文件。(别忘了把诸如-ED、-ER、-ERS、-S 的后缀附加到较短的字上。)现在取每个五字母单词 α，并把它的字母排为递增次序，得到有序的五个字母的序列 α'。最后把所有对偶(α', α)进行排序，来把所有变异单词弄到一起。

Kim. D. Gibson 于 1967 年所作的实验表明，普遍知道的五个字母变异字的第二个最长的集合为 LEAST，SLATE，STALE，STEAL，TAELS，TALES，TEALS。但如果他能够使用更大的辞典，那就能通过加上下列各词而使这个集合真正成为第一：ALETS(钢肩甲)，ASTEL(一个裂片)，ATLES(意愿)，LAETS(介于奴隶和自由民之间的人们)，LASET(白鼬)，LATES(尼罗河吻鲈)，LEATS(水沟)，SALET(中世纪的头盔)，SETAL(与刚毛有关的)，SLEAT(刺激、激励)，STELA(石柱)以及 TESLA(特斯拉，磁通量密度单位)。连同对于“settle”(安置)和“teasel”(川续断属植物)的旧的拼写 SATEL、TASEL 和 TASLE，我们得到 22 个互相可以转换的字，其中没有一个需要以大写字母来拼写。而且再凭一点勇气，我们还可以加上老的英文字 tœsl，德文的 atles，以及 de Staël 夫人！当我们参考完备的词典时，也能把集合{LAPSE，LEAPS，PALES，PEALS，PLEAS，SALEP，SPEAL}扩充到至少含 14 个字。[参见 H. E. Dudeney，由 Martin Gardner 主编，*300 Best Word Puzzles* (纽约：Chas. Scribner's Sons，1968)，智力游戏 190 和 194；Ross Eckler，*Making the Alphabet Dance* (纽约：St. Martin's Griffin，1997)，图 46c]。

三个或以上五个字母的英文变异字的开初的和末尾的集合是{ALBAS、BALAS、BASAL}以及{STRUT、STURT、TRUST},如果不允许包括适当的名字的话。然而,如果去掉该限制,则适当的名字导致一个较早的集合{ALBAN、BALAN、BANAL、LABAN、NABAL、NABLA}。在普通英语中较长的变异字的最令人惊异的也许是数学的集合{ALERTING(警戒)、ALTERING(修改)、INTEGER(整数)、RELATING(关联)、TRIANGLE(三角形)}。

更快地进行的方法是计算 $f(\alpha)=(h(a_1)+h(a_2)+\cdots+h(a_5))\bmod m$,其中 $a_1,\cdots,a_5$ 是 α 中个别字母的数值代码,且 $(h(1),h(2),\cdots)$ 是 26 个随机地选择的常数;这里 m 比如说是 $2^{\lfloor 2\lg N\rfloor}$,如果有 N 个单词。对文件 $(f(\alpha),\alpha)$ 用算法 5.2.5R 的两次扫描进行排序将把这些异体字放到一起;然后当 $f(\alpha)=f(\beta)$ 时,我们必须确保对于 $\alpha'=\beta'$ 有一个正确的变异字。$f(\alpha)$ 的值可以比 α' 更快地计算,而且对于文件中大多数的字 α 这个方法避免对 α' 的确定。

注意:当我们要把有相等的多字键码 $(a_1,\cdots,a_n)$ 的记录的所有集合弄到一起时,可以使用一个类似的技术。假设,除了把有相同键码的记录弄在一起外,我们并不关心文件的次序;有时,对一个字的键码 $(a_1x^{n-1}+a_2x^{n-2}+\cdots+a_n,)\bmod m$ 进行排序列,其中 x 是任意固定的值,而不是对原来多字的键码进行排序,还要更快些。

22. 求图形的同构不变量(即,在同构的有向图上取相等值的函数)并对这些不变量进行排序,来把"明显地不同构"的图形彼此分开。同构不变量的例子是:(a)以 (a_i,b_i) 表示顶点 v_i,这里 a_i 是它的输入次数,b_i 是它的输出次数;然后把对偶 (a_i,b_i) 排为字典编辑顺序。得到的文件是同构不变量。(b)用 (a_i,b_i,a_j,b_j) 表示从 v_i 到 v_j 的一条弧,并且把这四元组排成字典编辑顺序。(c)把有向图分开成为连通分量(参考算法 2.3.3E),确定每个分量的不变量,并且以某种方式把这些分量按它们的不变量的次序来排序。也见习题 21 中的讨论。

在按它们的不变量对有向图进行排序之后,一般仍然有必要进行第二次测试看是否具有相等不变量的有向图确是同构的。这些不变量对于这些测试也是有帮助的。在自由树的情况下,有可能找出"特征的"或"规范的"不变量,它们完全地表征树,使我们不必进行第二次测试[参见 J. Hopcroft 和 R. E. Tarjan, *Complexity of Computer Computations* (New York: Plenum, 1972), 140~142]。

23. 一个方法是形成包含所有三个人的集体的一个文件,然后把它变换为含有所有的四个人的集体的文件,等等;如果没有很大的集体,则这个方法将是十分令人满意的。(另一方面,如果有 n 个人的集体,则至少有 $\binom{n}{k}$ 个 k 个人的集体,所以甚至当 n 仅仅是 25 人左右时,这个方法仍可能失败。)

以 $(a_1,\cdots,a_{k-1})$ 的形式给出列举 k 个人的集体的一个文件,其中 $a_1<\cdots<a_{k-1}$,我们可以通过(i)对于使 $b<c$,其分别的形式为 $(a_1,\cdots,a_{k-2},b)$,$(a_1,\cdots,a_{k-2},c)$ 的每对 $k-1$ 个人的集体,建立包含项 $(b,c,a_1,\cdots,a_{k-2})$ 的一个新文件;(ii)按它的头两个分量对这个文件进行排序;(iii)在同原来给定的文件中的一对相

识者(b,c)匹配的新文件中的每个项$(b,c,a_1,\cdots,a_{k-2})$，输出 k 个人的集体$(a_1,\cdots,a_{k-2},b,c)$。

24.（由 Norman Hardy 给出的解，大约在 1967 年）作输入文件的另一个副本；按照头一个分量对一个副本进行排序，按第二个分量对另一个副本进行排序。顺序地扫描这些文件，即可对于 $1\leqslant i\leqslant n-2$ 建立含所有对偶(x_i,x_{i+2})的一个新文件，并标识(x_{N-1},x_N)。对偶$(N-1,x_{N-1})$和(N,x_N)应写到另一个文件上。

归纳地继续这个过程。假定对于 $1\leqslant i\leqslant N-t$，文件 F 以随机次序包含所有对偶(x_i,x_{i+t})，而且对于 $N-t<i\leqslant N$ 文件 G 以第二个分量的顺序，包含所有对偶(i,x_i)。设 H 是文件 F 的一个副本，而且通过头一个分量对 H 排序，通过第二分量对 F 排序。现在扫描 F,G 和 H，同时建立两个新文件 F' 和 G' 如下。如果文件 F,G,H 的当前记录分别为(x,x')，(y,y')，(z,z')则：

i)如果 $x'=z$，则输出(x,z')到 F'，且推进文件 F 和 H。

ii)如果 $x'=y'$，则输出$(y-t,x)$到 G'，且推进文件 F 和 G。

iii)如果 $x'>y'$，则推进文件 G。

iv)如果 $x'>z$，则推进文件 H。

当文件 F 被取尽时，通过第二个分量对 G' 排序而且把 G 同它合并；然后以 $2t$ 代替 t，以 F' 代替 F，以 G' 代替 G。

于是 t 取值 2,4,8,…；对于固定的 t 我们扫描数据 $O(\log N)$次来对它进行排序；因此扫描的总数是 $O((\log N)^2)$。最后 $t\geqslant N$，所以 F 是空的；然后我们只需按 G 的头一个分量对 G 进行排序即可。

25.（由 D. Shanks 给出的一个思想）对于 $0\leqslant n<m$，编制两个文件，一个包含 $a^{mn}\bmod p$，而另一个包含 $ba^{-n}\bmod p$。对这些文件进行排序而且寻找一个共同条目。

注意：[这使最坏情况运行时间从 $\Theta(p)$减少到 $\Theta(\sqrt{p}\log p)$。进一步的重大改进通常是可能的。例如，通过测试是否 $b^{(p-1)/2}\bmod p=1$ 或$(p-1)$，在 $\log p$ 步之内，我们能容易地确定 n 是偶数还是奇数。一般地说，如果 f 是 $p-1$ 的任何因子，而 d 是 $\gcd(f,n)$的任何因子，通过查找长度为 f/d 的一个表中 $b^{(p-1)/f}$的值，我们可类似地确定$(n/d)\bmod f$。如果 $p-1$ 有质因子 $q_1\leqslant q_2\leqslant\cdots\leqslant q_t$，而且如果 q_t 很小，因此，在对于基数 $q_1,\cdots,q_t$ 的混合进制表示中，通过从右到左地找出数字，我们能快速地计算 n。（这个思想归功于 R. L. Silver1964；也请参见 S. C. Pohlig 和 M. Hellman, *IEEE Transactions* **IT-24**(1978), 106～110。）

John M. Pollard 基于随机映射理论，发现了以大约 $O(\sqrt{p})$次运算 mod p，同时要求很少内存的计算离散的 log 的漂亮方法。请见*Math. Comp.*, **32**(1978), 918～924，其中他也建议了基于数 $n_j=r^j\bmod p$ 的另一个方法，这些数有很小的质因子。

习题 4.5.4-46 讨论了一些渐近地更快的方法。

5.1.1 小节

1.2 0 5 2 2 3 0 0 0;2 7 3 5 4 1 8 6。

2.$b_1=(m-1) \bmod n$;$b_{j+1}=(b_j+m-1) \bmod (n-j)$。

3.$\bar{a}_j=a_{n+1-j}$("反序的排列")。这一思想为 O. Terquem[*Journ. de Math.* **3** (1838),559～560]用来证明,在一个随机排列中反序的平均数是$\frac{1}{2}\binom{n}{2}$。

4.**C1**. 置 $x_0 \leftarrow 0$。(对于 $1 \leqslant j \leqslant n$,有可能让 x_j 在以下的诸步骤中同 b_j 共享存储)

C2. 对于 $k=n,n-1,\cdots,1$(以这一次序)进行下列工作:置 $j \leftarrow 0$;然后置 $j \leftarrow x_j$ 恰 b_k 次;然后置 $x_k \leftarrow x_j$ 且 $x_j \leftarrow k$。

C3. 置 $j \leftarrow 0$。

C4. 对于 $k=1,2,\cdots,n$(以这一次序)进行下列工作:置 $a_k \leftarrow x_j$,然后置 $j \leftarrow x_j$。▌

为节省存储空间,请见习题 5.2-12。

5. 命 α 是非负整数的有序对的一个串$[m_1,n_1]\cdots[m_k,n_k]$; $|\alpha|=k$,表示 α 的长度。命 ε 表示空串(长度为 0)。考虑在这种有序对上递归地定义的二元操作。如下:

$$\varepsilon \circ \alpha = \alpha \circ \varepsilon = \alpha;$$

$$([m,n]\alpha) \circ ([m',n']\beta) =$$

$$\begin{cases}[m,n](\alpha \circ [m'-m,n']\beta)), & \text{如果 } m \leqslant m' \\ [m',n']((([m-m'-1,n]\alpha) \circ \beta), & \text{如果 } m > m'\end{cases}$$

由此得出为求 $\alpha \circ \beta$ 的值所需要的计算时间与 $|\alpha \circ \beta|=|\alpha|+|\beta|$ 成比例。其次,我们可以证明$\circ$是可以结合的,而且$[b_1,1]\circ[b_2,2]\circ\cdots\circ[b_n,n]=[0,a_1][0,a_2]\cdots[0,a_n]$。左边的表达式可以在$\lceil \lg n \rceil$次扫描中计算出来,每次扫描都结合一些串对,总共为 $O(n \log n)$个步骤。

例子:从(2)开始,我们来求[2,1]∘[3,2]∘[6,3]∘[4,4]∘[0,5]∘[2,6]∘[2,7]∘[1,8]∘[0,9]的值。头一次扫描把它归结为[2,1][1,2]∘[4,4][1,3]∘[0,5][2,6]∘[1,8][0,7]∘[0,9]。第二次扫描把它归结为[2,1][1,2][1,4][1,3]∘[0,5][1,8][0,6][0,7]∘[0,9]。第三次扫描得到[0,5][1,1][0,8][0,2][0,6][0,4][0,7][0,3]∘[0,9]。第四次扫描得出(1)。

动机:一个诸如[4,4][1,3]的串表示"␣␣␣␣4␣3␣∞",这里"␣"表示一个空格;操作 $\alpha \circ \beta$ 把 β 的空格和非空格插入到 α 的空格中。注意,和习题 2 一起,我们得到 Joseph 问题的一个算法。它是 $O(n \log n)$,而不是 $O(mn)$,同时部分地回答了在习题 1.3.2-22 中提出的一个问题。

以一种直截了当的方式使用平衡树,即得到对于这个问题的另一个 $O(n \log n)$ 的解,这种解法使用一个随机存取的存储器。

6. 由 $b_1=b_2=\cdots=b_n=0$ 开始。对于 $k=\lfloor \lg n\rfloor,\lfloor \lg n\rfloor-1,\cdots,0$,进行如下:对于 $0\leqslant s\leqslant n/2^{k+1}$ 置 $x_s\leftarrow 0$,然后对于 $j=1,2,\cdots,n$ 进行如下:置 $r\leftarrow\lfloor a_j/2^k\rfloor \bmod 2$, $s\leftarrow\lfloor a_j/2^{k+1}\rfloor$(这些实质上是位的抽取);如果 $r=0$,则置 $b_{a_j}\leftarrow b_{a_j}+x_s$,如果 $r=1$ 则置 $x_s\leftarrow x_s+1$。

另一个答案出现于习题 5.2.4-21 当中。

7. $B_j<j$ 且 $C_j\leqslant n-j$,因为 a_j 左边有少于 $j-1$ 个元素,而右边有 $n-j$ 个元素。为了从 $B_1B_2\cdots B_n$ 重新构造 $a_1a_2\cdots a_n$,由元素 1 开始;然后对于 $k=2,\cdots,n$,对每个 $\geqslant k-B_k$ 的元素加 1 并附加 $k-B_k$ 于右边(参见 1.2.5 小节的方法 2)。对诸 C 的一个类似的过程有效。或者我们可以使用下列习题的结果。[Rodrigues,*J. de. Math.* **4** (1839),236~240 讨论了 c 反序表。C 反序表在 1800 年由 Rothe 使用,参见 Netto 的*Lehrbuch der Combinatorik* (1901) § 5]。

8. $b'=C,c'=B,B'=c,C'=b$,因为 $a_1\cdots a_n$ 的每对反序(a_i,a_j)对应于 $a'_1\cdots a'_n$ 的反序(j,i)。某些进一步的关系:(a)$c_j=j-1$ 当且仅当(对于所有的 $i<j$ 有 $b_i>b_j$);(b)$b_j=n-j$ 当且仅当(对于所有的 $i>j$ 有 $c_i>c_j$);(c)$b_j=0$ 当且仅当(对于所有的 $i>j$ 有 $c_i-i<c_j-j$);(d)$c_j=0$ 当且仅当(对于所有的 $i<j$ 有 $b_i+i<b_j+j$;(e)$b_i\leqslant b_{i+1}$当且仅当 $a'_i<a'_{i+1}$,当且仅当 $c_i\geqslant c_{i+1}$;(f)$a_j=j+C_j-B_j$;$a'_j=j+b_j-c_j$。

9. $b=C=b'$等价于 $a=a'$。

10. $\sqrt{10}$。(对截八面体建立坐标的一种方式是命下列向量$(1,0,0)$,$(0,1,0)$,$\frac{1}{2}(1,1,\sqrt{2})$,$\frac{1}{2}(1,-1,\sqrt{2})$,$\frac{1}{2}(-1,1,\sqrt{2})$,$\frac{1}{2}(-1,-1,\sqrt{2})$分别代表对偶 21,43,41,31,42,32 的相邻的交换。这些向量的和给出$(1,1,2\sqrt{2})$作为顶点 4321 与 1234 之间的差。)

一个更对称的解是通过

$$\sum\{\mathbf{e}_u-\mathbf{e}_v\mid(u,v)\text{是 }\pi\text{ 的一个反序}\}$$

来表示四维中的顶点 π,这里 $e_1=(1,0,0,0)$,$e_2=(0,1,0,0)$,$e_3=(0,0,1,0)$,$e_4=(0,0,0,1)$。于是 $1\ 2\ 3\ 4\leftrightarrow(0,0,0,0)$;$1\ 2\ 4\ 3\leftrightarrow(0,0,-1,1)$;$\cdots$;$4\ 3\ 2\ 1\leftrightarrow(-3,-1,1,3)$。所有的点都位于三维子空间$\{(w,x,y,z)\mid w+x+y+z=0\}$上;两个相邻顶点之间的距离为$\sqrt{2}$。等价地(参见习题 8(f))我们可以通过向量$(a'_1,a'_2,a'_3,a'_4)$表示 $\pi=a_1a_2a_3a_4$,这里 $a'_1a'_2a'_3a'_4$ 是逆排列。(这个以排列作为坐标的截八面体的 4 维表示,以及它的 n 维推广。C. Howard Hinton 在*The Fourth Dimension*(伦敦,1904)第 10 章中讨论过。许多年以后 Guilband 和 Rosenstiehl 发现了一些进一步的性质,他们把图 1 叫做“排列面体”;请见习题 12。)

把截八面体的一些复制以所谓“最简单的”方式充填三维空间[见 H. Steinhaus,*Mathematical Snapshots*(牛津,1960),200~203;C. S. Smith,*Scientific American* **190** (1954 年 1 月),58~64]。Pappus 的*Collection*(大约公元 300 年)提到截八面体作

为阿基米德研究的 13 个特殊的固体图形之一。阿基米德固体的图示即非棱形多面体有把任何顶点转换成任何其它顶点的对称性，而且它的面是正规多边形但不完全相等，可在比方说，由 H.S.M.Coxeter 修改了的 W.W.Rouse Ball 的书*Mathematical Recreations and Essays* (Macmillan,1939)，第 5 章；H.Martyn Cundy 和 A.P.Rollett, *Mathematical Models* (牛津,1952)，94～109 中找到。

11.(a)显然。(b)由诸顶点$(1,2,\cdots,n)$及如果 $x>y$ 且$(x,y)\in E$ 或$x<y$ 且$(y,x)\in \bar{E}$ 时由 $x\to y$ 形成的弧，构成一个有向图。如果没有有向回路，则这个有向图可以拓扑地进行排序，而且得到的线性次序是所希望的排列。如果有一个有向的回路，则它的长度至少为 3，因为没有长度为 1 或 2 的，而且由于一个更长的回路 $a_1\to a_2\to a_3\to a_4\to\cdots\to a_1$ 可以被缩短(或者 $a_1\to a_3$ 或者 $a_3\to a_1$)。但是长度为 3 的一条有向回路包含 E 或$\bar{E}$ 的两条弧，这证明了 E 或$\bar{E}$ 毕竟不是传递的。

12.[G.T.Guilbaud 和 P.Rosenstiehl,*Math et Sciences Humaines* **4** (1963),9～33]。假设$(a,b)\in\bar{E}$,$(b,c)\in\bar{E}$,$(a,c)\notin\bar{E}$。则对于某个 $k\geqslant 1$，我们有 $a=x_0>x_1>\cdots>x_k=c$，这里$(x_i,x_{i+1})\in E(\pi_1)\cup E(\pi_2)$，其中 $0\leqslant i<k$。考虑这种类型的一个反例，其中 k 为极小。由于$(a,b)\notin E(\pi_1)$和$(b,c)\notin E(\pi_1)$，我们有$(a,c)\notin E(\pi_1)$，而且类似地$(a,c)\notin E(\pi_2)$；因此 $k>1$。但是如果 $x_1>b$，则$(x_1,b)\in\bar{E}$，同 k 的极小性矛盾，而$(x_1,b)\in E$ 意味着$(a,b)\in E$。类似地如果 $x_1<b$ 则我们发现$(b,x_1)\in\bar{E}$ 和$(b,x_1)\in E$ 都是不可能的。

13. 对于反序表中 $b_1,\cdots,b_{m-1},b_{m+1},\cdots,b_n$ 的任何固定的选择。当 b_m 跑遍它所有可能的值 $0,1,\cdots,m-1$ 时，总和 $\Sigma_i b_i$ 将恰取每一 modulo m 剩余一次。

14. 题中提示的构造把不同部分的分划的对偶互相转换，两个情况 $j=k=p_k$ 和$j=k=p_k-1$ 除外。在例外的情况下，n 分别为$(2j-1)+\cdots+j=(3j^2-j)/2$ 及$(2j)+\cdots+(j+1)=(3j^2+j)/2$，而且有具有 j 个部分的一个惟一的不成对的分划[*Comptes Kendus Acad. Sci.* **81**(Paris,1881),448～450。Euler 原来的证明，在*Novi Comment. Acad. Sc. Pet.* **5** (1754),75～83 上，也是非常有趣的。他通过简单的运算证明：如果我们对于 $n\geqslant 1$，定义 s_n 为幂级数 $1-z^{2n-1}-z^{3n-1}s_{n+1}$，则无穷乘积等于 s_1。Knuth 和 Paterson 在*Fibonacci Quarterly* **16** (1978),198～212 上讨论了 Euler 无穷和的有限形式]。

15. 转置点的图式，以从诸 p 进行到诸 P。容易得到诸 P 的生成函数，因为我们首先选择任意数量的 1(生成函数 $1/(1-z)$)，然后独立地选择任意数量的 2(生成函数 $-1/(1-z^2)$)，$\cdots$，最后任意数量的 n。

16. 在头一个恒等式中 z^nq^m 的系数是把 m 分划成至多 n 部分的个数。在第二个恒等式中，它是把 m 分成 n 个不同的非负部分的个数：即形如 $m=p_1+p_2+\cdots+p_n$ 的和，其中 $p_1>p_2>\cdots>p_n\geqslant 0$。这和 $m-\binom{n}{2}=q_1+q_2+\cdots+q_n$ 相同，其中 $q_1\geqslant q_2\geqslant\cdots\geqslant q_n\geqslant 0$，对应关系为 $q_i=p_i-n+i$[*Commentarii Academiæ Scien-*

tiarum Petropolitanæ **13** (1741),64～93]。

注意:第二个恒等式是当 $n\to\infty$ 时,习题 1.2.6-58 的 q 多项式定理的极限。类似地,第一个恒等式,是在该习题的答案中证明的该定理的对偶形式的极限。

令 $n!_q=\prod_{k=1}^{n}(1+q+\cdots+q^{k-1})$,并令 $\exp_q(z)=\sum_{n=0}^{\infty}z^n/n!_q$。头一个恒等式告诉我们,当 $|q|<1$ 时,$\exp_q(z)$ 等于 $1/\prod_{k=0}^{\infty}(1-q^kz(1-q))$;第二个告诉我们,当 $|q|>1$ 时,它等于 $\prod_{k=0}^{\infty}(1+q^{-k}z(1-q^{-1}))$。得到的形式幂级数恒等式 $\exp_q(z)\exp_{q^{-1}}(-z)=1$ 等价于公式

$$\sum_{k=0}^{n}\frac{(-1)^kq^{k(k-1)/2}}{(1-q)\cdots(1-q^k)(1-q)\cdots(1-q^{n-k})}=\delta_{n0},\text{整数 } n\geqslant 0$$

它是对于 $x=-1$ 时 q 多项式定理的一个结果。

17.

0000	0100	0010	0001
1101	1201	1021	1012
1010	0110	0120	0102
1011	0111	0121	0112
1001	0101	0011	0012
2012	0212	0122	0123

18. 令 $q=1-p$。对于所有反序 α 的情况求和的和数 $\sum Pr(\alpha)$,也可以通过对 k 求和来计算,这里 $0\leqslant k<n$ 是具下列性质的最左边的二进位位置的精确个数,在这些位置中,i 和 j 之间有等式成立且对于 $i<j$ 在一个反序 $X_i\oplus i>X_j\oplus j$ 中的 X_i 和 X_j 之间也有等式成立。由此我们得到公式 $\sum_{0\leqslant k<n}2^k(p^2+q^2)^k(p^2 2^{n-k-1}2^{n-k-1}+2pq2^{n-k-1}(2^{n-k-1}-1))$;求和并简化,得到 $2^{n-1}(p(2-p)(2^n-(p^2+q^2)^n)/(2-p^2-q^2)+(p^2+q^2)^n-1)$。

19. 反序的个数是 $\sum_{0<i<j<n}(\lfloor mj/n\rfloor-\lfloor mi/n\rfloor-\lfloor m(j-i)/n\rfloor)=\sum_{0<i<j<n}[mj \bmod n<mi \bmod n]=\sum_{0<r<n}\lfloor mr/n\rfloor(r-(n-r)-(n-r-1))$,它可以被转换成 $\frac{1}{4}(n-1)(n-2)-\frac{1}{4}n\sigma(m,n,0)$。[*Crelle* **198** (1957).162～166。]

20. 请见 J. J. Sylvester,*Amer. J. Math.* **5**(1882),251～330,**6**(1883),334～336,§57～§68;E. M. Wright,*J. London Math. Soc.* **40** (1965),55～57;以及 J. Zolnowsky,*Discrete Math.* **9** (1974),293～298。

Jacobi 恒等式可以快速地证明如下:因为

$$\prod_{k=1}^{n}(1-u^kv^{k-1})=(-1)^nu^{\binom{n+1}{2}}v^{\binom{n}{2}}\prod_{k=1}^{n}(1-u^{-k}v^{1-k})$$

对于 $q=uv$ 习题 1.2.6-58 的 q 多项式定理告诉我们,

$$\prod_{k=1}^{n}(1-u^k v^{k-1})(1-u^{k-1}v^k)=(-1)^n u^{\binom{n+1}{2}} v^{\binom{n}{2}} \prod_{k=-n+1}^{n}(1-u^{k-1}v^k)=$$

$$(-1)^n u^{\binom{n+1}{2}} v^{\binom{n}{2}} \sum_j \binom{2n}{j}_{uv} (uv)^{\binom{j}{2}} (-u^{-n}v^{1-n})^j=$$

$$\sum_j \binom{2n}{n+j}_{uv} (-1)^j u^{\binom{j}{2}} v^{\binom{j+1}{2}}$$

两边乘以 $\prod_{k=1}^{n}(1-u^k v^k)=\prod_{k=1}^{n}(1-q^k)$，并且注意，对于固定的 j，我们有 $\binom{2n}{n+j}_q \prod_{k=1}^{n}(1-q^k)=1+O(q^{n+1-|j|})$。当 $n\to\infty$ 时，Jacobi 恒等式就得到了。

21. 把 C_j 解释成第 j 次输出后栈上的元素个数。（关于 b 的特征和栈排列的 B 表，请见习题 2.3.3-19)。

22. (a)把数 $\{1,2,\cdots,n\}$ 安排在一个圆周上就如同在一个钟的面上那样，并指向 1。然后对于 $j=n,n-1,\cdots,1$（以这个次序），以反时钟的方向移动指针 h_j+1 步，从这个圆周上删去被指的数并把它称为 a_j。

(b)统计每个 i，就像环绕 $a_i a_{i+1}\cdots a_n$ 那么频繁；这就是对于 $j\geqslant i$，$a_j>a_{j+1}$ 的次数。因此，对于 $a_j>a_{j+1}$ 的每个 j 对应于曾被统计过一次的下标 $1,\cdots,j$。[韩国牛，数学进展 **105**(1994)，28～29]；在下道题的上下文内，Rawlings 得到一个等价的结果]。

23. 例如假设 $n=5$ 和 $a_1\,a_2\,a_3\,a_4\,a_5=3\;1\;4\;2\;5$。对于某个非负整数的 k_j，在每个死者之前未射中的个数必定是 $2+5k_1$，$2+4k_2$，$1+3k_3$，$1+2k_2$，k_5。注意在上题的记号下，对偶的排列 14253 有 h 表 01122。一般地说，得到 $a_1a_2\cdots a_n$ 的概率将是

$$\sum_{k_1,\cdots,k_n\geqslant 0}(q_1^{h_n+nk_1}p_1)(q_2^{h_{n-1}+(n-1)k_2}p_2)\cdots(q_n^{h_1+k_n}p_n)=$$

$$\frac{1-q_1}{1-q_1^n}\,\frac{1-q_2}{1-q_2^{n-1}}\cdots\frac{1-q_n}{1-q_n^1}q_1^{h_n}q_2^{h_{n-1}}\cdots q_n^{h_1},$$

其中 $p_j=1-q_j$ 是在 $j-1$ 个死者之后致命的概率，而且 $h_1h_2\cdots h_n$ 对应于 $a_1a_2\cdots a_n$ 的对偶。特别是，当 $p_1=\cdots=p_n=p=1-q$ 时，这个概率是 $q^{h1+\cdots+hn}/G_n(q)$。因此最不可能的次序是 $n\cdots 21$。[J. Treadway 和 D. Rawlings，*Math. Mag.* **67** (1994)，345～354；Rawlings 在 *Int. J. Math. & Math. Sci.* **15** (1992)，291～312 上把这个过程推广到多重集合的排列中。]

24. 令 $a_0=0$，并且说，如果 $a_j>t(a_{j+1})$，则一个广义的下降出现在 $j<n$ 处。在 a_{j-1} 和 a_j 之间插入 n 引起一个新的广义的下降当且仅当 $a_{j-1}<t(a_j)<n$。假设当 j 有值 $j_1>j_2>\cdots>j_k>0$ 时出现；令 j 的其它值是 $j_n>j_{n-1}>\cdots>j_{k+1}$。于是 $j_n=n$，而且可以证明，当把 n 插在 a_{j_k} 紧前边时，广义的下标增加 $n-k$。[对于某个 $d\geqslant 0$，使 $t(j)=j+d$ 的特殊情况是由 D. Rawlings 给出的，见 *J. Combinatorial Theory*

A31 (1981), 175～183; *Linear and Multilinear Algebra* **10** (1981), 253～260 中他把这个特殊情况推广到多重集合排列中]。

本题对于排列定义 $n!$ 个不同的统计学，每个有出现于(7)和(8)中的生成函数 $G_n(z)$。我们通过如下产生俄国轮盘的方法，可定义许多这样的统计：在 $j-1$ 个死之后，开始下一轮射击的人是 $f_j(a_1,\cdots,a_{j-1})$，其中 f_j 是在 $\{1,\cdots,n\}\setminus\{a_1,\cdots,a_{j-1}\}$ 中取值的一个任意函数。[参见韩国牛, *Calcul Denertien* (Thesis, Univ. Stvasbourg, 1992), 1.3 部分, §7]。

25.(a)如果 $a_1<a_n$，$h(\alpha)$有和 α 一样多的反序，因为 α_j 的元素现在反序 x_j 而不是 a_n。但是如果 $a_1>a_n$，$h(\alpha)$的反序个数少 $n-1$ 个，因为 x_j 损失它的 a_n 的反序，以及 α_j 中每个元素的反序。因此，如果我们置 $x_n=a_n$，并且递归地令 $x_1\cdots x_{n-1}=f(h(\alpha))$，排列 $f(\alpha)=x_1\cdots x_n$ 有所要求的性质。我们有 $f(198263745)=912638745$ 以及 $f^{[-1]}(198263745)=192687345$。

(b)当 α^- 是 α 的逆时，关键之点是 $\operatorname{inv}(\alpha)=\operatorname{inv}(\alpha^-)$ 以及 $\operatorname{ind}(\alpha^-)=\operatorname{ind}(f(\alpha)^-)$。因此如果 $\alpha_1=\alpha^-$，$\alpha_2=f(\alpha_1)$，$\alpha_3=\alpha_2^-$，$\alpha_4=f^{[-1]}(\alpha_3)$以及 $\alpha_5=\alpha_4^-$，我们有

$$\operatorname{inv}(\alpha_5)=\operatorname{inv}(\alpha_4)=\operatorname{ind}(\alpha_3)=\operatorname{ind}(\alpha_2^-)=\operatorname{ind}(\alpha_1^-)=\operatorname{ind}(\alpha)$$
$$\operatorname{ind}(\alpha_5)=\operatorname{ind}(\alpha_4^-)=\operatorname{ind}(\alpha_3^-)=\operatorname{ind}(\alpha_2)=\operatorname{inv}(\alpha_1)=\operatorname{inv}(\alpha)$$

[*Math. Nachrichten* **83** (1978), 143～159]。

26.[由 Doron Zeilberger 提供的解] $\operatorname{inv}(\alpha)\operatorname{ind}(\alpha)$的平均是

$$\frac{1}{n!}\sum_{\alpha}\sum_{1\leqslant j<k\leqslant n}\quad\sum_{1\leqslant l<n}[a_j>a_k]\,l\,[a_l>a_{l+1}]$$

它是次数≤4 的 n 的一个多项式。对于 $1\leqslant n\leqslant 5$ 计算这个和给出分别的值 $0,\frac{1}{2}$，$\frac{6}{2},\frac{21}{2},\frac{55}{2}$；所以这个多项式必定是$\frac{1}{8}n(n-1)+\frac{1}{16}n^2(n-1)^2$。由(12)和(13)，对于 $n\geqslant 2$，减去 $\operatorname{mean}(g_n)^2$ 和除以 $\operatorname{var}(g_n)$给出答案 $9/(2n+5)$。

27.当把 $q_n\cdots q_2q_1$ 当作是一个多重集合的排列时，我们有 $\operatorname{inv}(a_1a_2\cdots a_n)=\operatorname{inv}(q_n\cdots q_2q_1)$(参见 5.1.2 小节)。由此得出，利用答案 16 的记号和习题 5.1.2-16 的结果

$$\frac{H_n(w,z)}{(1-z)\cdots(1-z^n)}=\sum_{a_1\cdots a_n}w^{\operatorname{inv}(a_1\cdots a_n)}z^{\operatorname{ind}(a_1\cdots a_n)}\sum_{p_1\geqslant\cdots\geqslant p_n\geqslant 0}z^{p_1+\cdots+p_n}=$$

$$\sum_{q_1,q_2,\cdots,q_n\geqslant 0}w^{\operatorname{inv}(q_n\cdots q_2q_1)}z^{q_1+q_2+\cdots+q_n}=$$

$$\sum_{k_0+k_1+k_2+\cdots=n}\binom{n}{k_0,k_1,k_2,\cdots}_w z^{k_1+2k_2+\cdots}=$$

$$n!_w[u^n]\sum_{k_0,k_1,k_2,\cdots}\prod_{j=0}^{\infty}\frac{(z^ju)^{k_j}}{k_j!_w}=$$

$$n!_w[u^n]\prod_{j=0}^{\infty}\exp_w(z^ju)=$$

$$n!_w[u^n]\prod_{j=0}^{\infty}\prod_{k=0}^{\infty}\frac{1}{1-z^jw^ku(1-w)},$$

于是我们有漂亮的恒等式

$$\prod_{j,k\geqslant 0}\frac{1}{1-w^jz^ku}=\sum_{n\geqslant 0}\frac{H_n(w,z)u^n}{(1-w)(1-w^2)\cdots(1-w^n)(1-z)(1-z^2)\cdots(1-z^n)}$$

这是由 D. P. Roselle 在*Proc. Amer. Math. Soc*, **45** (1974),144～150 上对生成函数 $H_n(w,z)=\sum_\alpha w^{\mathrm{ind}(\alpha^-)}z^{\mathrm{ind}(\alpha)}$建立的。习题 25 表明同样的双变量生成函数统计下标和反序。这里给出的证明是由 Garsia 和 Gessel 给出的[Advances in Math. **31** (1979),288～305],他们继续下去得到更为一般得多的结果。

在习题 4.7-27 中置 $m=\infty$导致递推式

$$H_n(w,z)=\sum_{k=1}^{n}\binom{n}{k}_w z^{n-k}\Big(\prod_{j=1}^{k-1}(1-z^{n-j})\Big)H_{n-k}(w,z)$$

28. 交换相邻的两个元素把总共的位移改变 0 或 ±2;因此 $\mathrm{td}(a_1a_2\cdots a_n)\leqslant 2\ \mathrm{inv}(a_1a_2\cdots a_n)$

我们也可证明 $\mathrm{td}(a_1a_2\cdots a_n)\geqslant\mathrm{inv}(a_1a_2\cdots a_n)$。假设 j 是离开位置的最小元素,且设 $a_k=j$。设 l 是使$l<k$ 和 $a_l\geqslant k$ 的极大。交换 a_l 和 a_k 使反序减少 $2(k-l)-1$,并使总共的位移减少 $2(k-l)$。因此如果为了对一个给定的排列 $a_1a_2\cdots a_n$ 进行排序需要 m 次重复这个算法,我们有 $\mathrm{td}(a_1a_2\cdots a_n)=\mathrm{inv}(a_1a_2\cdots a_n)+m$。

注意:一个随机排列的平均总位移是$(n^2-1)/3$;见习题 5.2.1-7。看起来对于总共的位移的生成函数没有一个简单的形式。

29. 我们可以作为 $\mathrm{inv}(\pi)$个转置 τ_j 的一个乘积得到π,其中 τ_j 交换j 和$j+1$。例如,图 1 中的通路 1234→1324→1342→3142 对应于 τ_2,然后 τ_3,然后 τ_1;因此 $3142=\tau_1\tau_3\tau_2$。因此通过作 $\mathrm{inv}(\pi)$转置,其中每一个转置使反序个数改变 ±1,$\pi\pi'$可从π'得到。由此得出,$\mathrm{inv}(\pi\pi')\leqslant\mathrm{inv}(\pi)+\mathrm{inv}(\pi')$。如果等式成立,每个转置增加一个新插入,因此 $E(\pi\pi')\supseteq E(\pi')$。

反之,如果 $E(\pi\pi')\supseteq E(\pi')$,我们要来证明$|E(\pi,\pi')|-|E(\pi')|=\mathrm{inv}(\pi\pi')-\mathrm{inv}(\pi')$个转置的某个序列将把 π'转换成$\pi\pi'$。这样的转置定义 π,所以这将证明 $\mathrm{inv}(\pi)\leqslant\mathrm{inv}(\pi\pi')-\mathrm{inv}(\pi')$;因此等式必定成立。例如,假设 $\pi'=314592587$ 和 $E(\pi\pi')\supseteq E(\pi')$。如果 $E(\pi\pi')$不包含(4,1),(5,4),(9,5),(6,2)或(8,6),则 $\pi\pi'$必然等于π',否则 $E(\pi\pi')$包含它们之一,比如说(9,5);于是 $E(\pi\pi')$包含 $E(\tau_4\pi')=E(314952687)$。这样,我们可以通过对$|E(\pi\pi')|-|E(\pi')|$利用归纳法来证明这

个结果。

5.1.2 小节

1. 假的,(由于一项相当重要的技术细节)。如果你说是“真的”,你大概不知道在 4.6.3 小节中给出的 $M_1 \cup M_2$ 的定义,它具有这样的性质:每当 M_1 和 M_2 为集合时,$M_1 \cup M_2$ 也是集合。实际上 $\alpha \top \beta$ 是 $M_1 \uplus M_2$ 的一个排列。

2. $b\ c\ a\ d\ d\ a\ d\ a\ d\ b$。

3. 肯定不是,因为我们可以有 $\alpha=\beta$。(然而,惟一因子分解定理证明,没有太多的可能性。)

4. $(d)\top(b\ c\ d)\top(b\ b\ c\ a\ d)\top(b\ a\ b\ c\ d)\top(d)$。

5. 对偶 $\cdots xx \cdots$ 出现的个数,等于 $\frac{x}{x}$ 的列数,减 0 或减 1。当 x 是最小的元素时,则当且仅当 x 不是排列中的头一个时,诸出现的个数相等。

6. 计算两行阵列的相关数很容易:$\binom{m}{k}\binom{n}{k}$。

7. 利用定理 B 的(a)部分,类似于(20)的推导给出

$$\binom{A-1}{A-k-m-1}\binom{B}{m}\binom{C}{k}\binom{B+k}{B-l}\binom{C-k}{l}$$
$$\binom{A-1}{A-k-m}\binom{B}{m}\binom{C}{k}\binom{B+k-1}{B-l-1}\binom{C-k}{l}$$
$$\binom{A-1}{A-k-m}\binom{B}{m}\binom{C}{k}\binom{B+k-1}{B-l}\binom{C-k}{l}.$$

8. 完全的质因子分解为 $(d)\top(b\ c\ d)\top(b)\top(a\ d\ b\ c)\top(a\ b)\top(b\ c\ d)\top(d)$,它是惟一的,因为没有相邻的对偶可交换。所以有 8 个解,且 $\alpha=\epsilon,(d),(d)\top(b\ c\ d),\cdots$。

10. 假的,但在一些有趣的情况下为真。给定质数的任何线性次序,至少有一个所述形式的因子分解,因为每当违反这个条件时,我们可以作一个交换来减少因子分解中的“反序”个数。所以条件不成立的原因仅仅是某些排列有多于一个这样的因子分解。

设 $\rho\sim\sigma$ 意味着 ρ 同 σ 可交换。对于所述的因子分解的惟一性来说,下列条件是必要和充分的:

$$\rho\sim\sigma\sim\tau \quad 且 \quad \rho\prec\sigma\prec\tau \quad 意味着\ \rho\sim\tau$$

证明:如果 $\rho\sim\sigma\sim\tau$ 和 $\rho\prec\sigma\prec\tau$ 且 $\rho\not\sim\tau$,则我们将有两个因子分解 $\sigma\top\tau\top\rho=\tau\top\rho\top\sigma$;因此条件是必要的。反之,为了证明它对于惟一性是充分的,令 $\rho_1\top\cdots\top\rho_n=\sigma_1\top\cdots\top\sigma_n$ 是满足条件的两个不相同的因子分解。我们可以假定 $\sigma_1\prec\rho_1$,因此对于某个 $k>1$ 有 $\sigma_1=\rho_k$;其次,对于 $1\leqslant j<k$,$\sigma_1\sim\rho_j$。由于 $\rho_{k-1}\sim\sigma_1=\rho_k$,我们有 $\rho_{k-1}\prec\sigma_1$;因此 $k>2$。设 j 使得 $\sigma_1\prec\rho_j$,而且对于 $j<i<k$,有 $\rho_i\prec\sigma_1$。则 $\rho_{j+1}\sim\sigma_1\sim\rho_j$ 和 $\rho_{j+1}\prec\sigma_1\prec\rho_j$ 意味着 $\rho_{j+1}\sim\rho_j$;因此 $\rho_j\prec\rho_{j+1}$,矛盾。

因此,若给定在质元素的一个集合 S 上的一个有序关系,它满足上述条件,而且如果知道一个排列 π 的所有质因子属于 S,则我们可以得出结论说 π 有所述类型的一个惟一的因子分解。例如,当 S 是(29)中循环的集合时,这样的条件成立。

但是,不能用这种办法对所有质元素的集合排序。因为如果有,比如说 $(a\ b)\prec(d\ e)$,则我们被迫定义

$$(a\ b)\prec(d\ e)\succ(b\ c)\prec(e\ a)\succ(c\ d)\prec(a\ b)\succ(d\ e)$$

矛盾。(也请看下一习题。)

11. 我们希望证明,如果 $p(1)\cdots p(t)$ 是 $(1,\cdots,t)$ 的一个排列,则当且仅当 $\sigma_{p(1)}\top\cdots\top\sigma_{p(t)}=\sigma_1\top\cdots\top\sigma_t$ 时排列 $x_{p(1)}\cdots x_{p(t)}$ 被拓扑地排序;而且如果 $x_{p(1)}\cdots x_{p(t)}$ 和 $x_{q(1)}\cdots x_{q(t)}$ 是不同的拓扑排序,则我们对于某个 j 有 $\sigma_{p(j)}\neq\sigma_{q(j)}$。第一个性质可由下列事实推出:$x_{p(1)}$ 成为一个拓扑排序中的头一个的充要条件是 $\sigma_{p(1)}$ 与 $\sigma_{p(1)-1}\cdots\sigma_1$ 可交换(但不同于它们);而且这个条件意味着 $\sigma_{p(2)}\top\cdots\top\sigma_{p(t)}=\sigma_1\top\cdots\top\sigma_{p(1)-1}\top\sigma_{p(1)+1}\top\cdots\top\sigma_t$,所以我们可以使用归纳法。第二个性质可由下列事实推出,如果 j 是使 $p(j)\neq q(j)$ 中之极小的,则由拓扑排序的定义,我们有,比如说 $p(j)<q(j)$ 以及 $x_{p(j)}\not\prec x_{q(j)}$;因此 $\sigma_{p(j)}$ 同 $\sigma_{q(j)}$ 没有公共的字母。

为了得到一个任意的偏序,设循环 σ_k 由所有使得 $x_i\prec x_j$ 且 $i=k$ 或 $j=k$ 的有序对偶 (i,j) 组成;这些有序对偶作为循环的个别元素以某种任意次序出现。于是对于偏序 $x_1\prec x_2$, $x_3\prec x_4$, $x_1\prec x_4$ 的循环将是 $\sigma_1=((1,2)(1,4))$, $\sigma_2=((1,2))$, $\sigma_3=((3,4))$, $\sigma_4=((1,4)(3,4))$。

12. 不能形成其它循环。因为,例如,原来的排列不包括 ^a_c 列。如果 $(a\ b\ c\ d)$ 出现 s 次,则 $(a\ b)$ 必然出现 $A-r-s$ 次,因为有 $A-r$ 个 ^a_b 列,而且仅仅两类循环贡献给这样的列。

13. 在两行的记号下,首先放置形如 ^d_a 的 $A-t$ 个列,然后放置其他的 t 个 a 于第二行中,然后放置诸 b,最后放置剩下的字母。

14. 由于在 π^- 的两行记号下,任意给定的字母下边的元素,是处于非减次序的,我们不总有 $(\pi^-)^-=\pi$;但是 $((\pi^-)^-)^-=\pi^-$ 是真的。事实上,恒等式

$$(\alpha\top\beta)^-=((\alpha^-\top\beta^-)^-)^-$$

对于所有的 α,β 成立。(见习题 5-2)

给定一多重集合,其不同的字母为 $x_1<\cdots<x_m$,注意到它们的每一个都有形如 $\beta_1\top\cdots\top\beta_m$ 的惟一质因子分解,这里 β_j 有 0 个或多个质因子 $(x_j)\top\cdots\top(x_j)\top(x_jx_{k_1})\top\cdots\top(x_jx_{k_t})$, $j<k_1\leqslant\cdots\leqslant k_t$,我们可以此表征它的自逆排列。例如,$(a)\top(a\ b)\top(a\ b)\top(b\ c)\top(c)$ 是一个自逆排列,$\{m\cdot a,n\cdot b\}$ 的自逆排列个数因此是 $\min(m,n)+1$;对于 $\{l\cdot a,m\cdot b,n\cdot c\}$ 的自逆排列个数是不等式 $x+y\leqslant l$, $x+z\leqslant m$, $y+z\leqslant n$ 的非负整数解 x,y,z 的个数。一个集合的自逆排列的个数在 5.1.4

节中考虑。

在两行记号下有 n_{ij} 个$\frac{x_i}{x_j}$的出现的$\{n_1\cdot x_1,\cdots,n_m\cdot x_m\}$的排列的个数是 $\Pi_i n_i!\,/\,\Pi_{i,j}n_{ij}!$，和在两行记号下有 n_{ij} 个$\frac{x_j}{x_i}$的出现的个数相同。因此应该有一个更好的定义一个多重集合排列的逆的方法。例如，如果像在定理 C 中一样，π 的质因子分解是 $\sigma_1\top\sigma_2\top\cdots\top\sigma_t$，则我们可定义 $\pi^- = \sigma_t^-\top\cdots\top\sigma_2^-\top\sigma_1^-$，其中$(x_1\cdots x_n)^- = (x_n\cdots x_1)$。

Dominique Foata 和韩国牛已经发现，如果以这样的方式，即 π 和 π^- 有相同个数的反序，来定义反序是更合乎要求的，因为给定数 n_{ij} 的反序的生成函数是 $\Pi_i n_i!_z/\Pi_{i,j}n_{ij}!_z$；见习题 16。然而，看起来这并不是定义有该性质的一个乘方的自然方式。

15. 见定理 2.3.4.2D，撤销该有向图的一条有向边，定能得到一棵有向树。

16. 如果 $x_1<x_2<\cdots$，对于诸 x_j 的反序表条款必定有 $b_{j1}\leqslant\cdots\leqslant b_{jn_j}$ 的形式。其中 b_{jn_j}(最右的 x_j 的反序的个数)至多是 $n_{j+1}+n_{j+2}+\cdots$。所以反序表的第 j 部分的生成函数分划成至多 n_j 个部分，没有超出 $n_{j+1}+n_{j+2}+\cdots$部分的生成函数。分划成至多 m 个部分，无超出 n 的部分的生成函数，是 z 多项式系数$\binom{m+n}{m}_z$；这通过归纳法很容易证明，而且它也可以借助于 F. Franklin 给出的巧妙的构造来证明。[*Amer. J. Math* . **5** (1882), 268～269；也请见 Pólya 和 Alexanderson, *Elemente dev Mathematik* **26** (1971), 102～109。]对于 $j=1,2,\cdots$把诸生成函数相乘给出多重集合的排列的反序所要求的公式。MacMahon 在*Proc. London Math. Soc.* (2) **15** (1916), 314～321 上发表了它。

17. 设 $h_n(z)=(n!_z)/n!$；则所求的概率生成函数是 $g(z)=h_n(z)/h_{n_1}(z)\cdot h_{n_2}(z)\cdots$。由等式 5.1.1-(12)，$h_n(z)$的平均值是$\frac{1}{2}\binom{n}{2}$，所以 g 的平均值是

$$\frac{1}{2}\left(\binom{n}{2}-\binom{n_1}{2}-\binom{n_2}{2}\cdots\right)=\frac{1}{4}(n^2-n_1^2-n_2^2-\cdots)=\frac{1}{2}\sum_{i<j}n_in_j$$

类似地，方差是

$$\frac{1}{72}(n(n-1)(2n+5)-n_1(n_1-1)(2n_1,+5)-\cdots)=$$

$$\frac{1}{36}(n^3-n_1^3-n_2^3-\cdots)+\frac{1}{24}(n^2-n_1^2-n_2^2-\cdots)$$

18. 是的；可以直截了当地推广习题 5.1.1-25 的构造。或者，通过构造 m 元组$(q_1,\cdots,q_m)$为一方和 n 元组的有序对$((a_1,\cdots,a_n),(p_1,\cdots,p_n))$为另一方这两者之间的一一对应，其中 q_j 是包含 n_j 个非负整数的一个多重集合，$a_1\cdots a_n$ 是$\{n_1\cdot 1,\cdots,n_m\cdot m\}$的一个排列且 $p_1\geqslant\cdots\geqslant p_n\geqslant 0$，我们可以推广 5.1.1-(14)后面的证明。

给定 q_j 的所有元素的下标 j，这个对应和以前一样来定义。它满足条件

$$\sum(q_1)+\cdots\sum(q_m)=\mathrm{ind}(a_1\cdots a_n)+(p_1+\cdots+p_n)$$

其中$\sum(q_i)$是 q_j 的元素之和。[关于在这个证明中所使用技术以及等式 5.1.3-(8)的推导进一步的推广，请见 D. E. Knuth, *Math. Comp.* **24** (1970), 955～961。也可参见由 Richard P. Stanley 在*Memoirs Amer. Math. Soc.* **119** (1972)中所作的广泛的讨论。]

19.(a)设 $S=\{\sigma|\sigma$ 是质元素，σ 是 π 的左因子$\}$。如果 S 有 k 个元素，π 的使 $\mu(\lambda)\neq0$ 的诸左因子 λ 恰巧是 S 子集的 2^k 个插入(请看定理 C 的证明)；因此 $\sum\mu(\lambda)=\Pi_{\sigma\in S}(1+\mu(\sigma))=0$，因为 $\mu(\sigma)=-1$ 而且 S 非空。(b)如果对于某个 $j\neq k$，有 $i_j=i_k$，则显然 $\epsilon(i_1\ldots i_n)=\mu(\pi)=0$。否则当 $i_1\cdots i_n$ 有 r 个反序时，$\epsilon(i_1\cdots i_n)=(-1)^r$；当 $i_1\cdots i_n$ 有 s 个偶循环时这就是$(-1)^s$；当 $i_1\cdots i_n$ 有 t 个循环时它是$(-1)^{n+t}$。

20.(a)根据插入的定义，显然。(b)根据定义

$$\det(b_{ij})=\sum_{1\leqslant i_1,\cdots,i_m\leqslant m}\epsilon(i_1\cdots i_m)b_{1i_1}\cdots b_{mi_m}$$

置 $b_{ij}=\delta_{ij}-a_{ij}x_j$，并应用习题 19(b)，我们得到

$$\sum_{n\geqslant0}\sum_{1\leqslant i_1,\cdots,i_n\leqslant m}x_{i_1}\cdots x_{i_n}\mu(x_{i_1}\cdots x_{i_n})\nu(x_{i_1}\cdots x_{i_n})$$

因为 $\mu(\pi)$通常为 0。

(c)当我们把诸 x 的乘积看作是不可交换变量的排列时，则利用自然的代数约定$(\alpha+\beta)\top\pi=\alpha\top\pi+\beta\top\pi$，可用习题 19(a)来证明 $D\top G=1$。

D. Zeilberger, *Discrete Math.* **56** (1985), 61～72 已经给出这个组合证明和其它重要定理的类似证明的一个简洁的表述。

21.对于 $k\leqslant0$，如果我们令 $n_k=0$，则为 $\Pi_{k=1}^{m}\binom{n_k+\cdots+n_{k-d}}{n_k}$，因为有 $\binom{n_m+\cdots+n_{m-d}}{n_m}$ 种方式来把诸 m 插入到$\{n_1\cdot1,\cdots,n_{m-1}\cdot(m-1)\}$的这样一个排列中。

22.(a)对于某个 k，$l(\pi)$的左右颠倒是在 $P_0(0^k1^{n_1}\cdots t^{n_t})$中；但代替颠倒 $l(\pi)$，我们将通过把 0 放在顶行之尾而不是之首给出它的两行形式。在 $l(\pi)$和 $r(\pi)$中 0 的个数 k 是对于$j\leqslant t<k$ 的 π 的两行形式中列 $\substack{j\\k}$ 的个数；这也是使 $k\leqslant t<j$ 的列的个数。从 $l(\pi)$和 $r(\pi)$的两行形式我们可以容易地重新构造 π，因为使 $j,k\leqslant t$ 的每个列$\substack{j\\k}$ 出现在$l(\pi)$中，使 $t<j,k$ 的每一列出现在$r(\pi)$中，剩下的列通过从左到右地把 $l(\pi)$的$\substack{j\\0}$ 或$\substack{0\\k}$ 同$r(\pi)$的$\substack{0\\k}$ 或$\substack{j\\0}$ 合并而得到。

(b)设 π 是所述形式的一个排列，并设 σ 是 $P_0(0^{n_0}1^{n_1}\cdots m^{n_m})$的任何排列。构造 λ 如下：删去 σ 的头 n_0 个项；然后以诸 x 代替诸 0，并以 π 的头 n_0 个项来作下

标;以诸 y 来代替其它元素,以 π 的剩下的非 0 项作为下标。并且还构造 ρ 如下:删去 σ 的诸 0,并且从左到右,根据 π 的列^j_k 有 $k=0$ 或 $k\neq 0$,用 x_j 或 y_j 来代替 j 的 n_j 个出现。例如,如果 $\pi=\begin{matrix}000000111112222333333\\231313023101020320 10\end{matrix}$,且 $\sigma=\begin{matrix}000000111112222333333\\32313201103201300201\end{matrix}$,我们有 $\lambda=x_2y_2y_3x_3y_1y_1x_1y_2y_3x_3x_1y_2x_3y_1$ 和 $\rho=y_3y_2y_3x_1x_3x_2y_1y_1y_3y_2y_1x_3x_2x_1$。反之,从 λ 和 ρ,我们可以重新构造 π 和 σ。

(c)我们在(a)的构造中有 $w(\pi)=w(l(\pi))w(r(\pi))$;因为 π 的列^j_k 或变成 $l(\pi)$ 或 $r(\pi)$ 中权 w_j/w_k 的^j_k,或者它被因子分解成有权 z_j/z_0 和 z_0/z_k 的列$^j_0$ 和^u_k。如果 $l(\pi)$ 有 p_j 个列^0_j 和 q_j 个列$^j_0$,它的权是 $\Pi^t_{j=1}(z_j^{q_j}w_j^{n_j-q_j}/z_j^{p_j}w_j^{n_j-p_j})=\Pi^t_{j=1}(w_j/z_j)p_jq_j$。现在 $\Pi^t_{j=1}(w_j/z_j)^{-q_j}$是 $\Pi^t_{j=1}(w_j/z_j)^{q_j}$的复共轭,所以对 $P_0(0^k1^{n_1}\cdots t^{n_t})$的所有元素权之和可简化为

$$\frac{k!(n_1+\cdots+n_t-k)!}{n_1!\cdots n_t!}\left|\sum_{p_1+\cdots+p_t=k}\binom{n_1}{p_1}\cdots\binom{n_t}{p_t}\left(\frac{w_1}{z_1}\right)^{p_1}\cdots\left(\frac{w_t}{z_t}\right)^{p_t}\right|^2$$

类似的说明对 $r(\pi)$也适用。所述之和是正的因为对于 $k=0$ 的项非 0。

23. 我们可以假定原来的线已被排序。设在上题的部分(c)中,$t=2$,$m=4$,$w_1=w_3=z_1=z_2=+1$,$w_2=w_4=z_3=z_4=-1$。于是 $w(\pi)=(-1)^d$,其中 d 是使 $j\neq k$ 的列^j_k 的个数。[见 Cillis 和 Zeilberger,*European J. Comb.* **4** (1983),221～223。这个结果首先由 Askey、Ismail 和 Koornwinder,*J. Comb. Theory.* **A 25** (1978),277～287 以完全不同的方法证明的。他们发现了在多重集合的排列和(Laguerre)多项式 $L_n^a(x)=\sum_{k=0}^n\binom{n+a}{n-k}(-x)^k/k!$ 的乘积的积分之间有趣的联系]。对于五个字母的字母表类似的结果为假,因为{1,2,3,4,5}的 5! 排列包括有偶数差的 1+10+45,和包括有奇数差的 0+20+44。

24. (a)把$\begin{matrix}w & x\\y & z\end{matrix}$转置两次恢复成$\begin{matrix}w & x\\y & z\end{matrix}$。给定排序$\begin{pmatrix}x_1\cdots x_n\\y_1\cdots y_n\end{pmatrix}=\begin{pmatrix}x''_1\cdots x''_n\\y''_1\cdots y''_n\end{pmatrix}$,通过找出在顶上的行中最左的 x 并把它转置到左边来使之无序。这引出适当的 y。(排序$\begin{pmatrix}x'_2\cdots x'_n\\y'_2\cdots y'_n\end{pmatrix}$的值也是惟一确定的)。

(b)我们实质上把 π 的两行记号表达成形式

$$\pi=\text{排序}\begin{pmatrix}y_1\cdots x_{1n}\,y_2\cdots x_{2n_2}\cdots y_t\cdots x_{tn_t}\\x_{11}\quad y_1\,x_{21}\cdots y_2\cdots x_{t1}\cdots y_t\end{pmatrix}$$

而且部分(a)为我们提供了我们精确地需要的工具。[当 R 保持两行记号的某些统计学时,这个构造提供了一些有趣的定理的组合证明。参见韩国牛,数学进展 **105** (1994),26～41。]

5.1.3 小节

1. 我们只需证明对于 $x=k$,当 $k\geqslant 1$ 时,这个值使(11)成立。利用(7),这个公

式变成为

$$k^n = \sum_{r=0}^{k} \left\langle {n \atop r-1} \right\rangle \binom{k+n-r}{n} =$$

$$\sum_{0\leqslant j\leqslant r\leqslant k} (-1)^j (r-j)^n \binom{n+1}{j}\binom{n+k-r}{n} =$$

$$\sum_{s=0}^{k} s^n \sum_{j=0}^{k-s} (-1)^j \binom{n+1}{j}\binom{n+k-s-j}{n}$$

当 $s<k$ 时，对 j 的求和可扩展到范围 $0\leqslant j\leqslant n+1$ 上，而且它为 0（j 的 n 次多项式的第 $n+1$ 次差分）。

2.(a)由习题 1.2.6-64 包含元素$(1,2,\cdots,q)$的每一个至少一次的序列 $a_1a_2\cdots a_n$ 的个数为 $\left\{ {n \atop q} \right\} q!$；对于 $m=q$，满足类似于(10)的式子的这类序列的个数为 $\binom{n-k}{n-q}$，因为我们必须选择 $n-q$ 个可能的＝符号。(b)对 $m=n-q$ 和 $m=n-q-1$ 把(a)的结果相加。

3.由(20)

$$\sum_n \frac{x^n}{n!}\sum_k \left\langle {n \atop k} \right\rangle (-1)^k = \frac{2}{e^{-2x}+1} = \frac{1}{x}\left(\frac{(-4x)}{e^{-4x}-1} - \frac{(-2x)}{e^{-2x}-1}\right) =$$

$$\frac{1}{x}\sum_{n\geqslant 0}\frac{B_n x^n}{n!}((-4)^n-(-2)^n)$$

因此结果是 $(-1)^{n+1}B_{n+1}2^{n+1}(2^{n+1}-1)/(n+1)$。或者，当 n 为奇数时，恒等式 $2/(e^{-2x}+1)=1+\tanh x$ 可让我们把答案表示成 $(-1)^{(n-1)/2}T_n$，这里 T_n 表示由公式

$$\tan z = T_1 z + T_3 z^3/3! + T_5 z^5/5! + \cdots$$

所定义的正切数。当 $n>0$ 为偶数时，由(7)，这和显然为 0。

碰巧，(18)现在产生新奇的 Stirling 恒等式

$$\sum_k \left\{ {n \atop k} \right\} \frac{k!}{(-2)^k} = \frac{2B_{n+1}(1-2^{n+1})}{n+1}$$

4.$(-1)^{n+m}\left\langle {n \atop m} \right\rangle$（考虑(18)中 z^{m+1} 的系数）。

5.通过公式(13)，习题 1.2.6-10，以及定理 1.2.4F，对于 $0\leqslant k<p$，$\left\langle {p \atop k} \right\rangle \equiv (k+1)^p - k^p \equiv (k+1)-k\equiv 1$ (modulo p)。

6.首先对 k 求和是不允许的，因为对于任意大的 j 和 k，这些项非 0，因此绝对值之和是无穷大。

作为较简单的出错例子，设 $a_{jk}=(k-j)[|j-k|=1]$，于是

$$\sum_{j\geqslant 0}\left(\sum_{k\geqslant 0} a_{jk}\right) = \sum_{j\geqslant 0}(\delta_{j0}) = +1 \text{ 而 } \sum_{k\geqslant 0}\left(\sum_{j\geqslant 0} a_{jk}\right) = \sum_{k\geqslant 0}(-\delta_{k0}) = -1$$

7. 是的。[F. N David 和 D. E. Barton, *Combinatorial Chance* (1962), 150～154；也参见习题 25 的答案。]

8. [(*Combinatory Analysis*) **1**(1915), 190]由容斥原理。例如，$1/(l_1+l_2)!\ l_3!\ (l_4+l_5+l_6)!$ 是 $x_1<\cdots<x_{l_1+l_2}, x_{l_1+l_2+1}<\cdots<x_{l_1+l_2+l_3}$ 以及 $x_{l_1+l_2+l_3+1}<\cdots<x_{l_1+l_2+l_3+l_4+l_5+l_6}$ 的概率。

N. G. de Bruijn 已经给出了用来计算其路段长度分别为$(l_1,\cdots,l_k)$的$\{1,\cdots,n\}$的排列之个数的一个简单的 $O(n^2)$ 算法，*Nieuw Archief voor Wiskunde* (3) **18** (1970), 61～65。

9. 在(23)中 $p_{km}=q_{km}-q_{k(m+1)}$。因为$\sum_{k,m}q_{km}z^m x^k=\frac{x}{1-x}g(x,z)$和 $g(x,0)=1$，我们有

$$h(z,x)=\sum h_k(z)x^k=\frac{x}{1-x}g(x,z)(1-z^{-1})+\frac{x}{1-x}z^{-1}=\frac{(1-z^{-1})x}{e^{(x-1)z}-x}+\frac{z^{-1}x}{1-x}$$

于是 $h_1(z)=e^z-(e^z-1)/z$；$h_2(z)=(e^{2z}-ze^z)+e^2-(e^{2z}-1)/z$。

10. 设 $M_n=L_1+\cdots+L_n$ 是均值；则$\sum M_n x^n=h'(1,x)$这里是对 z 求导数的，比如说这是 $x/(e^{x-1}-x)-x/(1-x)=M(x)$，由残数定理，如果我们沿着半径为 r 的一个圆周积分

$$\frac{1}{2\pi i}\oint M(z)z^{-n-1}dz = M_n-2\left(n+\frac{1}{3}\right)+1+\frac{z_1^{-n}}{z_1-1}+\frac{\bar{z}_1^{-n}}{\bar{z}_1-1}$$

其中$|z_1|<r<|z_2|$。(注意在 $z=1$ 处的二重极点)。其次，这个积分的绝对值小于$\oint|M(z)|r^{-n-1}dz=O(r^{-n})$。在越来越大的圆周上积分就给出收敛的级数 $M_n=2n-\frac{1}{3}+\sum_{k\geqslant 1}2\Re(1/z_k^n(1-z_k))$

为了确定方差，我们有 $h''(1,x)=-2h'(1,x)-2x(x-1)e^{x-1}/(e^{x-1}-x)^2$。一个类似于已对均值使用过的论证(但这次是对三重极点)证明了 $h''(1,x)$的系数渐近于 $4n^2+\frac{4}{3}n-2M_n$ 加些较小的项；由此得方差的渐近公式$\frac{2}{3}n+\frac{2}{9}$(加上指数较小的项)。

11. $P_{kn}=\sum_{t_1\geqslant 1,\cdots,t_{k-1}\geqslant 1}D(t_1,\cdots,t_{k-1},n,1)$，其中 $D(l_1,l_2,\cdots,l_k)$是习题 8 的 MacMahon 行列式。按它的头一行计算这个行列式，我们求出 $P_{kn}=c_0P_{(k-1)n}+c_1P_{(k-2)n}+\cdots+c_{k-2}P_{1n}-E_k(n)$，其中 c_j 和E_k 定义如下：

$$c_j=(-1)^j\sum_{t_1,\cdots,t_{j+1}\geqslant 1}\frac{1}{(t_1+\cdots+t_{j+1})!}=(-1)^j\sum_{m\geqslant 0}\binom{m}{j}\frac{1}{(m+1)!}=$$

$$(-1)^j\sum_{r,m\geqslant 0}\binom{-1}{j-r}\binom{m+1}{r}\frac{1}{(m+1)!}=-1+e\left(\frac{1}{0!}-\frac{1}{1!}+\cdots+(-1)^j\frac{1}{j!}\right)$$

$$E_1(n)=1/(n+1)!-1/n!;\ E_2(n)=1/(n+1)!;$$

$$E_k(n) = (-1)^k \sum_{m\geqslant 0} \binom{m}{k-3} \frac{1}{(n+2+m)!}, \quad k \geqslant 3$$

设 $P_{0n}=0, C(z)=\sum c_j z^j=(\mathrm{e}^{1-z}-1)/(1-z)$,并设

$$E(z,x) = \sum_{n,k} E_{k+1}(n) z^n x^k = 1 - \mathrm{e}^z + \frac{(\mathrm{e}^{1-x}-1)x^2}{(1-x)^2} - \frac{x^2(\mathrm{e}^{1-x}-\mathrm{e}^z)}{(1-x)(1-x-z)} + \frac{\mathrm{e}^z - 1 - z}{z(1-x)}$$

我们已经导出的递推关系等价于公式 $C(x)H(z,x)=H(z,x)/x+E(z,x)$;因此 $H(z,x)=E(z,x)x(1-x)/(x\mathrm{e}^{1-x}-1)$。展开这个幂级数给出 $H_1(z)=h_1(z)$(见习题 9);$H_2(z)=\mathrm{e}h_1(z)+1-\mathrm{e}^z$。

[注:头三个路段的生成函数是由 Knuth 导出的,*CACM* **6** (1963),685~688。Barton 和 Mallows,在*Ann. Math. Statistics* **36** (1965),249 页上指出了对于 $n\geqslant 1$ 的公式 $1-H_{n+1}(z)=(1-H_n(z))/(1-z)-L_n h_1(z)$和公式(25)。解决这个问题的另一个方法在习题 23 中说明。由于相邻的路段是不独立的,在这里所解决的问题和习题 9 的更简单(但大概更有用)的结果之间,没有简单的关系。]

12.[*Combinatory Analysis* **1** (1915),209~211] 把多重集合放置到 t 个可区别的盒子中的方式数为

$$N_t = \binom{t+n_1-1}{n_1}\binom{t+n_2-1}{n_2}\cdots\binom{t+n_m-1}{n_m}$$

因为有$\binom{t+n_1-1}{n_1}$种方法来放置诸 1,等等。如果要求没有空的盒子,则容斥原理告诉我们方式数为

$$M_t = N_t - \binom{t}{1}N_{t-1} + \binom{t}{2}N_{t-2} - \cdots$$

设 P_k 是有 k 个路段的排列数;如果我们在诸路段之间放置 $k-1$ 条垂直的线,并且在 $n-k$ 个剩下的位置上的任何地方放置 $t-k$ 条另外的垂直线,则得到把多重集合分成 t 个非空的可区别部分的 M_t 个方法之一。因此

$$M_t = P_t + \binom{n-t+1}{1}P_{t-1} + \binom{n-t+2}{2}P_{t-2} + \cdots$$

等置两个 M_t 的值,就可借助于 $N_1,N_2,\cdots$逐次地确定 $P_1,P_2,\cdots$。(我们希望见到一个更直接的证明。)

13. $1+\frac{1}{2}13\times 3=20.5$。

14. 由 Foata 对应式,给定的排列对应于

$$(31)\top(1)\top\cdots\top(4) = \begin{pmatrix}1&1&1&1&2&2&2&2&3&3&3&3&4&4&4&4\\3&1&1&2&3&4&3&2&1&1&3&4&2&2&4&4\end{pmatrix}$$

由(33)式这对应于

$$\begin{pmatrix}1&1&1&1&2&2&2&2&3&3&3&3&4&4&4&4\\2&4&4&3&3&3&1&1&4&4&2&1&2&1&2&3\end{pmatrix}$$

这对应于具有 9 个路段的 2 3 4 2 3 4 1 4 2 1 4 3 2 1 3 1。

15. 交替路段的个数是 1 加上使得 $1<j<n$ 且 $a_{j-1}<a_j>a_{j+1}$ 或 $a_{j-1}>a_j<a_{j+1}$ 的 j 的个数。对于固定的 j，这个概率为 $\frac{2}{3}$；因此对于 $n\geqslant 2$，平均是 $1+\frac{2}{3}(n-2)$。

16. 当新的元素 n 被插入到所有可能位置时，$\{1,2,\cdots,n-1\}$ 上的每个有 k 个交替路段的排列产生带有 k 个这样路段的 k 个排列，带有 $k+1$ 个这样路段的 2 个排列，以及带有 $k+2$ 个路段的 $n-k-2$ 个排列，因此

$$\left\langle\!\!\left\langle {n \atop k} \right\rangle\!\!\right\rangle = k\left\langle\!\!\left\langle {n-1 \atop k} \right\rangle\!\!\right\rangle + 2\left\langle\!\!\left\langle {n-1 \atop k-1} \right\rangle\!\!\right\rangle + (n-k)\left\langle\!\!\left\langle {n-1 \atop k-2} \right\rangle\!\!\right\rangle$$

命 $\left\langle\!\!\left\langle {1 \atop k} \right\rangle\!\!\right\rangle = \delta_{k0}$，$G_1(z)=1$ 是方便的。于是

$$G_n(z) = \frac{z}{n}((1-z^2)G'_{n-1}(z) + (2+(n-2)z)G_{n-1}(z))$$

微分导出 $x_n = G'_n(1)$ 的递推式

$$x_n = \frac{1}{n}((n-2)x_{n-1} + 2n - 2)$$

而这对于 $n\geqslant 2$ 有解 $x_n = \frac{2}{3}n - \frac{1}{3}$。另一个微分导致对于 $y_n = G''_n(1)$ 的递推式

$$y_n = \frac{1}{n}\left((n-4)y_{n-1} + \frac{8}{3}n^2 - \frac{26}{3}n + 6\right)$$

置 $y_n = \alpha n^2 + \beta n + \gamma$ 并对 α,β,γ 求解，对于 $n\geqslant 4$ 得到 $y_n = \frac{4}{9}n^2 - \frac{14}{15}n + \frac{11}{90}$。因此 $\mathrm{var}(g_n) = \frac{1}{90}(16n-29)$，$n\geqslant 4$。

这些均值和方差的公式是 J. Bienaymé 给出的，他未加证明地指出了它们[*Bull. Soc. Math. de Frauce* **2** (1874)，153～154；*Comptes, Rendus Acad. Sci.* **81** (Paris，1875)，417～423，也请看 458 页上 Bertrand 的注记]。$\left\langle\!\!\left\langle {n \atop k} \right\rangle\!\!\right\rangle$ 的递推关系是 D. André 给出的[*Comptes Rendus Acad. Sci.* **97** (Paris，1883)，1356～1358；*Annales Scientifiques. de l'École Normale Superieure* (3) **1** (1884)，121～134]。André 说明对于 $n\geqslant 4$，$g_n(-1)=0$；因此，具有偶数个交替路段的排列的个数是 $n!/2$。他也证明了均值的公式，并确定了具有极大的交替路段个数的排列的数目(见习题 5.1.4-23)。可以证明

$$G_n(z) = \left(\frac{1+z}{2}\right)^{n-1}(1+w)^{n+1}g_n\left(\frac{1-w}{1+w}\right); \quad w = \sqrt{\frac{1-z}{1+z}}, n\geqslant 2$$

其中 $g_n(z)$ 是递增路段的生成函数(18)。[见 David 和 Barton，*Combinatorial Chance*

（伦敦:Griffin,1962),157～162]。

17. $\binom{n+1}{2k-1}$;$\binom{n}{2k-2}$个以 0 结尾,$\binom{n}{2k-1}$个以 1 结尾。

18.(a)设给定的序列是如同在 5.1.1 节中的一个反序表。如果它有 k 个递降的路段,则对应的排列的反序中也有 k 个递降,(见答案 5.1.1-8(e));因此答案是 $\left\langle {n \atop k} \right\rangle$。(b)这个量满足 $f(n,k)=kf(n-1,k)+(n-k+1)f(n-1,k-1)$,因此它必定是$\left\langle {n \atop k-1} \right\rangle$。[参见 D. Dumont, *Duke Math. J.* **41** (1974). 313～315。]

19.(a)由定理 5.1.2B 的对应结果得$\left\langle {n \atop k} \right\rangle$。(b)有$(n-k)!$ 种方式把另外的 $n-k$ 个不相拼的车放置到整个棋盘上;因此答案是 $1/(n-k)!$ 乘$\sum_{j\geqslant 0} a_{nj}\binom{j}{k}$,其中由部分($a$),$a_{nj}=\left\langle {n \atop j} \right\rangle$。由习题 2,这导出$\left\{ {n \atop n-k} \right\}$。[Riordan 的*Introduction to Combinatorial Analysis* (Wiley,1958)的第 5 章一般地讨论了车的设置问题]。

由(E. A. Bender)给出的关于这个结果的一个直接证明,把使$\{1,2,\cdots,n\}$分成为 k 个不相交的非空子集的每一分划与$(n-k)$个车的一种排列联系起来:设分划是

$$\{1,2,\cdots,n\} = \{a_{11},a_{12},\cdots,a_{1n_1}\} \cup \cdots \cup \{a_{k1},a_{k2},\cdots,a_{kn_k}\}$$

其中对于 $1\leqslant j\leqslant n_i, 1\leqslant i\leqslant k$ 有 $a_{ij}<a_{i(j+1)}$。对于 $1\leqslant j\leqslant n_i, 1\leqslant i\leqslant k$,对应的排列把这些车放置到 $a_{i(j+1)}$行 a_{ij}列中。例如,图 4 中所示图形对应于分划$\{1,3,8\}\cup\{2\}\cup\{4,6\}\cup\{5\}\cup\{7\}$。

20. 阅读的次数是逆排列中路段的个数。头一个路段对应于头一次阅读,等等。

21. 它有 $n+1-k$ 个路段并需要 $n+1-j$ 次阅读。

22. [*J. Combinatorial Theory* **1** (1966),350～374]　如果 $rs<n$,则某次阅读将挑出 $t>r$ 个元素,$a_{i_1}=j+1,\cdots,a_{i_t}=j+t$,其中 $i_1<\cdots<i_t$。对于在 $i_k\leqslant m\leqslant i_{k+1}$范围内的所有 m,我们不可能有 $a_m>a_{m+1}$,所以这个排列至少在 $t-1$ 个位置上有 $a_m<a_{m+1}$;因此它至多有 $n-t+1$ 个路段。

另一方面,考虑排列 $\alpha_r\cdots\alpha_2\alpha_1$,其中块区 α_j 以递减次序包含数$\equiv j$(modulo r);例如,当 $n=9$ 和 $r=4$ 时,这个排列是 847362951。如果 $n\geqslant 2r-1$,这个排列有 $r-1$ 个递增,所以它有 $n+1-r$ 个路段。而且,如果 $r>1$,它恰好要求 $n+1-\lceil n/r\rceil$次阅读。我们可以任意地重新安排$\{kr+1,\cdots,kr+r\}$的元素而不改变路段的个数;这样,我们可以把阅读的次数减少到$\geqslant\lceil n/r\rceil$的任何所要求的值。

现在假设 $rs\geqslant n$ 和$r+s\leqslant n+1$,以及 $r,s\geqslant 1$。由习题 20 和 21,我们可以假定 $r\leqslant s$,因为有 $n+1-r$ 个路段和s 个阅读的一个排列的反序的反射有$n+1-s$ 个路段和 r 个阅读。于是上一段中的构造处理了除 $s>n+1-\lceil n/r\rceil$和 $r\geqslant 2$ 之外的所

有情况。为了完成证明,我们可以使用形如

$$2k+1\ 2k-1\cdots 1\ n+2-r\quad n+1-r\cdots 2k+2\quad 2k\cdots 2\ n+3-r\cdots n-1\ n$$

的一个排列,对于 $0\leqslant k\leqslant\frac{1}{2}(n-r)$,它有 $n+1-r$ 个路段和 $n+1-r-k$ 个阅读。

23. [*SIAM Review* **3** (1967),121~122] 假设无穷排列由取自于一致分布的独立样品组成。设 $f_k(x)\mathrm{d}x$ 是第 k 个最长的以 x 开始的概率,而且设 $g(u,x)\mathrm{d}x$ 是当上一个长路段以 u 开始时后一个长路段以 x 开始的概率。因此有 $f_1(x)=1$, $f_{k+1}(x)=\int_0^1 f_k(u)g(u,x)\mathrm{d}u$。我们有 $g(u,x)=\sum_{m\geqslant1}g_m(u,x)$,其中

$g_m(u,x)=\Pr(u<X_1<\cdots<X_m>x$ 或 $u>X_1>\cdots>X_m<x)=\Pr(u<X_1<\cdots<X_m)+\Pr(u>X_1>\cdots>X_m)-\Pr(u<X_1<\cdots<X_m<x)-\Pr(u>X_1>\cdots>X_m>x)=(u^m+(1-u)^m+|u-x|^m)/m!$

因此 $g(u,x)=\mathrm{e}^u+\mathrm{e}^{1-u}-1-\mathrm{e}^{|u-x|}$,而且求出 $f_2(x)=2\mathrm{e}-1-\mathrm{e}^x-\mathrm{e}^{1-x}$。可以证明 $f_k(x)$ 趋近于极限值 $\left(2\cos\left(x-\frac{1}{2}\right)-\sin\frac{1}{2}-\cos\frac{1}{2}\right)\Big/\left(3\sin\frac{1}{2}-\cos\frac{1}{2}\right)$。以 x 开始的一个路段的平均长度是 $\mathrm{e}^x+\mathrm{e}^{1-x}-1$;因此第 k 个长的路段的长度 $\mathscr{L}_k$ 是 $\int_0^1 f_k(x)(\mathrm{e}^x+\mathrm{e}^{1-x}-1)\mathrm{d}x$;$\mathscr{L}_1=2\mathrm{e}-3\approx2.43656$;$\mathscr{L}_2=3\mathrm{e}^2-8\mathrm{e}+2\approx2.42091$。类似的结果见 5.4.1 节。

24. 如同前面一样进行论证,结果是

$$1+\sum_{0\leqslant k<n}2^k(p^2+q^2)^k(p^2+2pq(2^{n-k-1}-1+q^2((2pq)^{n-k-1}-1)/(2pq-1)))$$

求和,并简化之得出

$$2^n(p^2+q^2)^n\left(p(p-q)/(p^2+q^2-pq)-\frac{1}{2}\right)+(2pq)^n pq^3/(p^2+q^2)\times$$
$$(p^2+q^2-pq)+q^2/(p^2+q^2)+2^{n-1}$$

25. 令 $V_j=(U_1+\cdots+U_j)\bmod 1$;则 $V_1,\cdots,V_n$ 是在[0..1)中独立一致的随机数,并且当且仅当 $\lfloor U_1+\cdots+U_n\rfloor=k$ 时形成有 k 个递降的一个排列。因此,答案是 $\left\langle {n \atop k} \right\rangle\Big/n!$,这是首先由 S. Tanny[*Duke Math. J*, **40** (1973),717~722]首先注意到的一个性质。

26. 例如,$\theta^5(1-z)^{-1}=(z-26z^2+66z^3+26z^4+z^5)/(1-z)^6$。

27. 下列规则定义了一个一一对应,它把有 k 个递降的一个排列 $a_1a_2\cdots a_n$ 对应到有 $k+1$ 个叶的一个 n 节点递增的森林:头一个根是 a_1,它的后裔是对应于 $a_2\cdots a_k$ 的森林,其中 k 是使得 $a_{k+1}<a_1$ 的极小值或者 $k=n$。[R. P. Stanley, *Enumerative Combinatorics* **1** (Wadsworth,1986),命题 1.3.16]。

28. $L(z)$的极是 $T(1/\mathrm{e})$的诸值,其中 $T(z)$是由 $T(z)=z\mathrm{e}^{T(z)}$所定义的(多值)树函数。于是对于 $m>0$,我们有收敛级数

$$z_M = -\sigma_m + \sum_{n\geqslant 0}\frac{1}{\sigma_m^n}\sum_k(-1)^k\left[{n \atop k}\right]\frac{(\ln\sigma_m)^{n+1-k}}{(n+1-k)!}, \sigma_m = -1-(2m+1)\pi i$$

[Corless, Gonnet, Hare, Jeffrey 及 Knuth,〈计算数学进展〉**5** (1996), 329～359, 公式(4.18)。]特别是,我们有 $z_m = (2m+\frac{1}{2}\pi i + \ln(2\pi e m) + \left(\frac{1}{4}-\frac{i}{2\pi}\right)\ln(2\pi e m))/m + O((\log m)^2/m^2)$。

令 $P(z) = \sum_{m=0}^{\infty}(z/(z-z_m) + z/(z\cdot\bar{z}_m))$,由此得出,对于 $x>1$, $P(x)-P(-x) = \sum_{m=0}^{\infty}4\Re(xz_m/(x^2-z_m^2)) = \sum_{M=1}^{\infty}O((x\ \log\ m)/(x^2+m^2)) = \sum_{m=1}^{x}O((x\ \log\ x)/x^2) + \sum_{m=x+1}^{\infty}O((x\ \log\ m)/m^2) = O(\log\ x)$。但我们知道,对于某个 c, $L(x)+P(x)=cx$;因此 $2cx = L(x)-L(-x)+O(\log\ x)$,而且通过在(25)中含 $x\to\infty$,我们求得 $c=-1/2$,因此 $L_1 = \sum_{m=0}^{\infty}zr_m^{-1}\cos\theta_m - 1/2$,(这一结果是由 Svante Janson 给出的)。

5.1.4 小节

1.

1	2	3	8
4	5	7	
6	9		

1	3	5	8
2	4	9	
6	7		

$$\begin{pmatrix}1&3&4&5&7&8&9\\5&9&2&4&8&1&7\end{pmatrix}$$

2. 当把 p_i 插入到列 t 中时,设列 $t-1$ 中的元素是 p_j。则 (q_j, p_j) 是在类 $t-1$ 中, $q_j<q_i$,而且 $p_j<p_i$;所以由归纳法,存在具有这个性质的下标 $i_1,\cdots,i_t$。反之,如果 $q_j<q_i$ 和 $p_j<p_i$ 而且如果 (q_j, p_j) 在类 $t-1$ 中,则当插入 p_i 时列 $t-1$ 包含一个 $<p_i$ 的元素,所以 (q_i, p_i) 在 $\geqslant t$ 的类中。

3. 当插入 p_i 时,这些列是"冲撞序列"(9)。行 1 和 2 反映了对于行 1 的操作,参见(14)。如果我们撤销其行 2 有∞的项的一些列,则如同在(15)中那样,行 0 和行 2 组成撞下的阵列。从行 k 进行到行 $k+1$ 的所述方法,恰是正文中确定类的算法。

4. (a)对图表的大小用归纳法,利用一个例子分析,并首先考虑对于行 1 的影响,然后考虑对于由行 1 撞下来的元素序列的影响。(b)可允许的交换可以模拟算法 I 的操作,且把图表表示成在这个算法之前和之后的一个规范排列。例如,通过一个可允许的交换序列(参考(4)和(5)),我们可以把

17 11 4 13 14 2 6 10 15 1 3 5 9 12 16 8

变换成为

17 11 13 4 10 14 2 6 9 15 1 3 5 8 12 16

5. 可允许的交换是左右对称的,而且当颠倒插入次序时, P 的规范排列显然地成为 P^T。

6. 设共有 t 类;这些类中恰有 k 个有奇数个元素,因为一个类中的元素的形式为

$$(p_{i_k},p_{i_1}),\quad (p_{i_{k-1}},p_{i_2}),\quad \cdots,\quad (p_{i_1},p_{i_k})$$

(见(18)和(22))。撞下的两行阵列恰有 $t-k$ 个不动点,这是由于构造它的方法所致;因此,由归纳法,这个图表减去它的头一行后,有 $t-k$ 个奇数长度的列。所以在头一行中的 t 个元素导致整个图表中 k 个奇数长度的列。

7.列的数目,即行 1 的长度,是类的数目(习题 2)。行数是 P^T 的列数,所以习题 5(或定理 D)完成了这个证明。

8.对于多于 n^2 个元素,对应的 P 图表必须或者多于 n 行或者多于 n 列。但有 $n\times n$ 的图表[这一结果最初是在*Compositio Math* **2** (1935),463~470 中证明的]。

9.这样的排列同形如$(n,n,\cdots,n)$的图表的对偶有一一对应关系,所以由(34),答案为$\left(\dfrac{n^2!\ \Delta(2n-1,2n-2,\cdots,n)}{(2n-1)!\ (2n-2)!\ \cdots n!}\right)^2=\left(\dfrac{n^2!}{(2n-1)(2n-2)^2\cdots n^n(n-1)^{n-1}\cdots 1^1}\right)^2$。对于这个问题,存在这样一个简单公式,真是惊人。我们也可以计算没有长于 m 的递增子序列和没有长于 n 的递减子序列的$\{1,2,\cdots,mn\}$的排列数。

10.我们归纳地证明在步骤 S3 中,$P_{(r-1)s}$和$P_{r(s-1)}$都小于 $P_{(r+1)s}$和$P_{r(s+1)}$。

11.我们当然也需要知道原来曾是 P_{11}的元素。然后有可能利用颇类似于算法 S 的一个算法,来进行恢复。

12.$\binom{n_1+1}{2}+\binom{n_2+2}{2}+\cdots+\binom{n_m+m}{2}-\binom{m+1}{3}$,即总共的遍历距离。极小值是习题 1.2.4-41 的序列 1,2,2,3,3,3,4,4,4,4…的头 n 项之和;这个和近似于$\sqrt{8/9}\,n^{3/2}$。(按照习题 29,在 n 个元素上的几乎所有图表都相当接近这个下限,所以平均次数是 $\Theta(n^{3/2})$)。

13.假设排列的元素是$\{1,2,\cdots,n\}$,因而 $a_i=1$;并且假设 $a_j=2$。情况 1,$j<i$。则 1 撞下 2,所以对应于 $a_1\cdots a_{i-1}a_{i+1}\cdots a_n$ 的图表的行 1 是 P^S 的行 1;而且撞下的排列除了它的最小元素 2 外,是以前被撞下的排列,所以我们可以对 n 用归纳法。情况 2,$j>i$。对 P^T 应用情况 1,并利用习题 5 及$(P^T)^S=(P^S)^T$ 这一事实。

15.如同在(37)中那样,例中的排列对应于图表

1	2	5	9	11
3	6	7		
4	8	10		

因此此数为 $f(l,m,n)=(l+m+n)!\ (l-m+1)(l-n+2)(m-n+1)/(l+2)!\ (m+1)!\ (n)!$,当然,假设 $l\geqslant m\geqslant n$。

16.由定理 H,80080。

17.由于 g 对诸 x 是反对称的,当 $x_i=x_j$ 时它为 0,所以对于所有的 $i<j$ 它可以被 x_i-x_j 所整除。因此 $g(x_1,\cdots,x_n;y)=h(x_1,\cdots,x_n;y)\Delta(x_1,\cdots,x_n)$。这里 h 必须是总次数为 1 的对 $x_1,\cdots,x_n,y$ 齐次的,而且对于 $x_1,\cdots,x_n$ 是对称的;所以对于仅仅依赖于 n 的某个a,b,$h(x_1,\cdots,x_n;y)=a(x_1+\cdots+x_n)+by$。我们通过

置 $y=0$ 可以计算 a；通过对 y 求偏微商，然后置 $y=0$ 可以计算 b。我们有

$$\frac{\partial}{\partial y}\Delta(x_1,\cdots,x_i+y,\cdots,x_n)\big|_{y=0}=\frac{\partial}{\partial x_i}\Delta(x_1,\cdots,x_n)=\Delta(x_1,\cdots,x_n)\sum_{j\neq i}\frac{1}{x_i-x_j}$$

最后

$$\sum_i\sum_{j\neq i}(x_i/(x_i-x_j))=\sum_i\sum_{j<i}(x_i/(x_i-x_j)+x_j/(x_j-x_i))=\binom{n}{2}$$

18. 它必须是 $\Delta(x_1,\cdots,x_n)\cdot(b_0+b_1y+\cdots+b_my^m)$，其中每个 b_k 是诸 x 的次数为 $m-k$ 的齐次对称多项式。我们有

$$\frac{\partial^k}{k!\partial y^k}\Delta(x_1,\cdots,x_i+y,\cdots,x_n)\big|_{y=0}=\Delta(x_1,\cdots,x_n)\sum\Bigl(1/\prod\nolimits_{l=1}^{k}(x_i-x_{j_l})\Bigr)$$

对于不同下标 $j_1,\cdots,j_k\neq i$ 的所有 $\binom{n-1}{k}$ 种选择进行求和。现在在 $b_k=\sum x_i^m/\prod_{l=1}^{k}(x_i-x_{j_l})$ 中，我们可以组合那些有一个给定的下标集合 $\{i,j_1,\cdots,j_k\}$ 的 $k+1$ 项的组；例如，当 $k=2$ 时，我们组合形如 $a^m/(a-b)(a-c)+b^m/(b-a)(b-c)+c^m/(c-a)(c-b)$ 的三项的集合。每个这样的组的和，由习题 1.2.3-33，是 $[z^{m-k}]\,1/(1-x_iz)(1-x_{j_1}z)\cdots(1-x_{j_k}z)$。我们因此求得

$$b_k=\sum_j\binom{n-j}{k+1-j}\sum s(p_1,\cdots,p_j)$$

其中 $s(p_1,\cdots,p_j)$ 是对于不同的下标 $i_1,\cdots,i_j\in\{1,\cdots,n\}$，由形如 $x_{i_1}^{p_1}\cdots x_{i_j}^{p_j}$ 的所有不同项所组成的单项对称函数；而内部的和是对于把 $m-k$ 分成恰好 j 个部分的所有分划进行的，即 $p_1\geqslant\cdots\geqslant p_j\geqslant 1$，$p_1+\cdots+p_j=m-k$。（这个结果是作者同 E. A. Bender 一起于 1969 得到的）。

当 $m=2$ 时答案是 $\Bigl(s(2)+(n-1)s(1)y+\binom{n}{3}y^2\Bigr)\Delta(x_1,\cdots,x_n)$；对于 $m=3$ 我们得到 $\Bigl(s(3)+((n-1)s(2)+s(1,1))y+\binom{n-1}{2}s(1)y^2+\binom{n}{4}y^3\Bigr)\Delta(x_1,\cdots,x_n)$；等等。

另一个表达式，作为

$$\Bigl(\binom{n}{k+1}z^k-\binom{n-1}{k+1}e_1z^{k+1}+\binom{n-2}{k+1}e_2z^{k+2}-\cdots\Bigr)/(1-e_1z+e_2z^2-\cdots)$$

中 z^m 的系数给出 b_k，其中 $e_l=\sum_{1\leqslant i_1<\cdots<i_l\leqslant n}x_{i_1}\cdots x_{i_l}$ 是一个初等对称函数。乘以 y^k 并对 k 进行求和即给出答案，它是

$$\frac{1}{yz}\Bigl(\frac{(1+z(y-x_1))\cdots(1+z(y-x_n))}{(1-zx_1)\cdots(1-zx_n)}-1\Bigr)\Delta(x_1,\cdots,x_n)$$

中 z^m 的系数。

19. 设转置图表的形状是 $(n_1',n_2',\cdots,n_r')$；答案是

$$\frac{1}{2}f(n_1,n_2,\cdots,n_m)\left(\frac{(\sum n_i^2-\sum n'^2_j)}{n(n-1)}+1\right)$$

其中 $n=\sum_{n_i}=\sum n'_j$。(利用关系式 $\sum in_i=\frac{1}{2}(n+\sum n'^2_j)$,这个公式可以被表达成不大对称的形式)。

注意:W. Feit[*Proc. Amer. Math. Soc*, **4** (1953),740～744] 证明,把整数$\{1,2,\cdots,n\}$放置到作为两个图表形状$(n_1,\cdots,n_m)\setminus(l_1,\cdots,l_m)$之"差"的一个数组中,其中 $0\leqslant l_j\leqslant n_j$ 和 $n=\sum(n_j-l_j)$,其方式个数为 $n!\ \det(1/((n_j-j)-(l_i-i))!)$。

20.在定理 H 之后的讨论中靠不住的论证,实际上对于这一情况是正确的(对应的概率都是独立的)。

注意:如果我们考虑对节点进行标号的所有 $n!$ 种方式,这里所考虑的标号是没有"反序"的那些。当树只不过是一条通路时这一特殊情况下,排列中的反序和树标号中的反序相同。参见 A. Björner 和 M.L. Wachs, *J. Combinatorial Theory* **A52** (1989),165～187。

21.[*Michigan Math. J.* **1** (1952),81～88] 设 $g(n_1,\cdots,n_m)=(n_1+\cdots+n_m)!\ \Delta(n_1,\cdots,n_m)/n_1!\ \cdots n_m!\ \sigma(n_1,\cdots,n_m)$,其中 $\sigma(x_1,\cdots,x_n)=\prod_{1\leqslant i<j\leqslant m}(x_i+x_j)$。为证明 $g(n_1,\cdots,n_m)$是填充移位图表的方式个数,我们必须证明 $g(n_1,\cdots,n_m)=g(n_1-1,\cdots,n_m)+\cdots+g(n_1,\cdots,n_m-1)$。对应于习题 17 的这个恒等式是 $x_1\Delta(x_1+y,\cdots,x_n)/\sigma(x_1+y,\cdots,x_n)+\cdots+x_n\Delta(x_1,\cdots,x_n+y)/\sigma(x_1,\cdots,x_n+y)=(x_1+\cdots+x_n)\Delta(x_1,\cdots,x_n)/\sigma(x_1,\cdots,x_n)$,同 y 无关;因为如果像在习题 17 中那样计算微商,则我们求得 $2x_ix_j/(x_j^2-x_i^2)+2x_jx_i/(x_i^2-x_j^2)=0$。

22.假设 $m=N$,必要时把一些 0 加到这个图形上;如果 $m>N$ 和 $n_m>0$,则方式数显然为 0。当 $m=N$ 时,这个答案是

$$\det\begin{pmatrix}\binom{n_1+m-1}{m-1} & \binom{n_2+m-2}{m-1} & \cdots & \binom{n_m}{m-1}\\ \vdots & \vdots & & \vdots\\ \binom{n_1+m-1}{0} & \binom{n_2+m-1}{0} & \cdots & \binom{n_m}{0}\end{pmatrix}$$

证明 我们可以假定 $n_m=0$,因为如果 $n_m>0$ 则这个阵列的头 n_m 列的第 i 行必然填的是 i,而且我们可以考虑剩下的图形$(n_1-n_m,\cdots,n_m-n_m)$。通过对 m 用归纳法,方式数成为

$$\sum_{\substack{n_2\leqslant k_1\leqslant n_1\\ \vdots\\ n_m\leqslant k_{m-1}\leqslant n_{m-1}}}\det\begin{pmatrix}\binom{k_1+m-2}{m-2} & \binom{k_2+m-3}{m-2} & \cdots & \binom{k_{m-1}}{m-2}\\ \vdots & \vdots & & \vdots\\ \binom{k_1+m-2}{0} & \binom{k_2+m-3}{0} & \cdots & \binom{k_{m-1}}{0}\end{pmatrix}$$

其中 n_j-k_j 表示在行 j 中诸 m 的个数。对于每个 k_j 的求和可以独立地进行，由此得

$$\det\begin{pmatrix} \binom{n_1+m-1}{m-1}-\binom{n_2+m-2}{m-1} & \binom{n_2+m-2}{m-1}-\binom{n_3+m-3}{m-1} & \cdots & \binom{n_{m-1}+1}{m-1}-\binom{n_m}{m-1} \\ \vdots & \vdots & & \vdots \\ \binom{n_1+m-1}{1}-\binom{n_2+m-2}{1} & \binom{n_2+m-2}{1}-\binom{n_3+m-3}{1} & \cdots & \binom{n_{m-1}+1}{1}-\binom{n_m}{1} \end{pmatrix}$$

即是所求的答案，因为 $n_m=0$。通过行运算，这个答案可以转化为一个 Vandermonde 行列式，同时给出公式 $\Delta(n_1+m-1,n_2+m-2,\cdots,n_m)/(m-1)!\ (m-2)!\ \cdots 0!$。[这一习题的答案，同群论中一个等价的问题相联系，它出现在 D.E. Littlewood 的*Theory of Group Characters* 牛津，1940)，189 中。]

23. [*Journalde Math*. (3) **7** (1881)，167～184.]（这是对于长度为 2 的所有路段，习题5.1.3-8的一种特殊情况，除开最后一个路段可能有长度 1 外。）当 $n\geqslant 2$ 时，元素 n 必出现在一行的最右边的位置之一上；一旦已经把它放到第 k 行最右边的框中，我们便有 $\binom{n-1}{2k-1}A_{2k-1}A_{n-2k}$ 种方式来完成这一工作。设

$$h(z)=\sum_{n\geqslant 1}A_{2n-1}z^{2n-1}/(2n-1)!=$$

$$\frac{1}{2}(g(z)-g(-z))$$

则

$$h(z)g(z)=\sum_{k,n\geqslant 1}\binom{n}{2k-1}A_{2k-1}A_{n-2k+1}z^n/n!=\Big(\sum_{n\geqslant 1}A_{n+1}z^n/n!\Big)-1=$$

$$g'(z)-1$$

以 $-z$ 代替 z 并且相加，得到 $h(z)^2=h'(z)-1$；因此，$h(z)=\tan z$。置 $k(z)=g(z)-h(z)$，我们有 $h(z)k(z)=k'(z)$；因此 $k(z)=\sec z$ 和 $g(z)=\sec z+\tan z=\tan\left(\frac{1}{2}z+\frac{1}{4}\pi\right)$。因此系数 A_{2n} 是欧拉数 $|E_{2n}|$；系数 A_{2n-1} 是正切数 $T_{2n-1}=(-1)^{n-1}4^n(4^n-1)B_{2n}/(2n)$。这些数的表出现于 Math. Comp. **21** (1967)，663～688 中。这个序列以 $(A_0,A_1,A_2,\cdots)=(1,1,1,2,5,16,61,272,1385,7936,\cdots)$ 开始。计算正切数和欧拉数最容易的方法大概是构造三角数组

$$\begin{array}{ccccccccccccc} &&&&&&1&&&&&& \\ &&&&&0&&1&&&&& \\ &&&&1&&1&&0&&&& \\ &&&0&&1&&2&&2&&& \\ &&5&&5&&4&&2&&0&& \\ &0&&5&&10&&14&&16&&16& \\ 61&&61&&56&&46&&32&&16&&0 \end{array}$$

其中部分和是交替地从左到右和从右到左形成的[A. J. Kempner, *Tôhoku Math. J.* **37** (1933), 348～349]。

25. 一般地说，如果 u_{nk} 是没有长度 $>k$ 的循环的$\{1,2,\cdots,n\}$上的排列的个数，则 $\sum u_{nk}z^n/n!=\exp(z+z^2/2+\cdots+z^k/k)$；这可以通过 $\exp(z)\times\cdots\times\exp(z^k/k)$ 来证明，并且得到

$$\sum_n z^n\Bigl(\sum_{j_1+2j_2+\cdots+kj_k=n} 1/1^{j_1}j_1!2^{j_2}j_2!\cdots\Bigr);$$

也可参见习题 1.3.2-21。类似地，$\exp(\sum_{s\in S}z^s/s)$ 是对于这样的排列的生成函数，这些排列的循环的长度全都是一个给定集合 S 的元素。

26. 由伽马函数积分，从 0 到∞的的积分是 $n^{(t+1)/4}\Gamma((t+1)/2)/2^{(t+3)/2}$（见习题 1.2.5-20，$t=2x^2/\sqrt{n}$）。所以从 $-\infty$ 到 ∞，当 t 为奇数时我们得到 0，否则 $n^{(t+1)/4}\sqrt{\pi}t!\ /2^{(3t+1)/2}(t/2)!$。

27. (a) 如果 $r_i<r_{i+1}$ 和 $c_i<c_{i+1}$，则条件 $i<Qr_ic_{i+1}<i+1$ 是不可能的。如果 $r_i\geqslant r_{i+1}$ 和 $c_i\geqslant c_{i+1}$，我们肯定不能有 $i+1\leqslant Q_{r_ic_{i+1}}\leqslant i$。(b) 通过对于 $a_1\cdots a_i$ 的图表中的行数用归纳法证明，$a_i<a_{i+1}$ 意味着 $c_i<c_{i+1}$，且 $a_i>a_{i+1}$ 意味着 $c_i\geqslant c_{i+1}$。(考虑行 1 和“撞下”的序列。(c) 这从定理 D(c) 得出。

28. 这一结果是 A. M. Vershik 和 S. V. Kerov 给出的，见*Dokl. Akad. Nauk SSSR* **233** (1977), 1024～1028，也可见 B. F. Logan 和 L. A. Shepp, *Advances in Math.* **26** (1977), 206～222。[J. Baik, P. Deift 及 K. Johansson, *AMM.* **12** (1999), 1119～1178 证明了标准离差是 $\Theta<n^{1/6}$]；而且，长度小于 $2\sqrt{n}+tn^{1/6}$ 的概率趋近 $\exp(-\int_t^\infty(x-t)u^2(x)dx)$，其中 $u''(x)=2u^3(x)+xu(x)$，且当 $x\to\infty$ 时 $u(x)$ 近似于函数 $A_i(x)$。]

29. $\binom{n}{l}\Big/l!$ 是长度为 l 的递增子序列的平均个数。(由习题 8 和 29，最大的递增子序列有长度 $\geqslant e\sqrt{n}$ 或 $\leqslant\sqrt{n}/e$ 的概率为 $O\left(1/\sqrt{n}\right)$)；[J. D. Dixon, *Discrete Math.* **12** (1975) 139～142]。

30. [*Discrete Math.* **2** (1972), 73～94; Marc van Leeuwen, Eleetronic J. *Combinatorics* **3**, 2(1996) 论文 #R15 已经给出了一个简化了的证明。]

31. $x_n=a_{\lfloor n/2\rfloor}$，其中 $a_0=1$，$a_1=2$，$a_n=2a_{n-1}+(2n-2)a_{n-2}$；$\sum a_nz^n/n!=\exp(2z+z^2)=(\sum t_nz^n/n!)^2$；对于偶数的 n，$x_n\approx\exp\Bigl(\frac{1}{4}n\ln n-\frac{1}{4}n+\sqrt{n}-\frac{1}{2}-$

$\frac{1}{2}\ln 2\Big)$。[参见 E. Lucas, *Théorie des Nombres* (1891), 217～223。]

32. 设 $m_n = \int_{-\infty}^{\infty} t^n e^{-(t-1)^2/2} dt / \sqrt{2\pi}$。则 $m_0 = m_1 = 1$，且如果我们进行分部积分，$m_{n+1} - m_n = nm_{n-1}$。所以由(40)，$m_n = t_n$。

33. 真的；它是 $\det_{i,j=1}^{m}\binom{a_i}{j-1}$ [Mitchell 在 *Amer. J. Math* **4** (1881), 341～344 中证明，它是现在称为一个 Schur(舒尔)函数的某个对称函数的展开中的项数。确实，如果 $0<a_1<\cdots<a_m$，它是 $S_{n_1n_2\cdots n_m}(x_1,x_2,\cdots,x_m)$ 中的项数，其中 $n_1=a_m-m$，$n_2=a_{m-1}-(m-1),\cdots,n_m=a_1-1$。这个舒尔函数是形状为$(n_1,\cdots,n_m)$的所有推广的图表之和，且在图表中的元素是对于所有的 j，作为 x_j 的乘积$\{1,\cdots,m\}$中的数，而推广的图表和通常的图表类似，但在行中允许有相同的元素。在这个定义中，我们允许参数 n_k 为 0。例如，$S_{210}(x_1,x_2,x_3)=x_1^2x_2+x_1^2x_3+x_1x_2^2+x_1x_2x_3+x_1x_2x_3+x_1x_3^2+x_2^2x_3+x_2x_3^2$，这是由于推广的图表 $\begin{smallmatrix}11\\2\end{smallmatrix}, \begin{smallmatrix}11\\3\end{smallmatrix}, \begin{smallmatrix}12\\2\end{smallmatrix}, \begin{smallmatrix}12\\3\end{smallmatrix}, \begin{smallmatrix}13\\2\end{smallmatrix}, \begin{smallmatrix}13\\3\end{smallmatrix}, \begin{smallmatrix}22\\3\end{smallmatrix}, \begin{smallmatrix}23\\3\end{smallmatrix}$ 所致。这样的图表的个数是 $\Delta(1,3,5)/\Delta(1,2,3)=8$。把算法 I 和 D 推广到推广的图表[*Pacific J. Math.*, **34** (1970), 709～727]，我们可以得到著名的恒等式

$$\sum_{\lambda} S_\lambda(x_1,\cdots,x_m)S_\lambda(y_1,\cdots,y_n) = \prod_{i=1}^{m}\prod_{j=1}^{n}\frac{1}{1-x_iy_j}$$

$$\sum_{\lambda} S_\lambda(x_1,\cdots,x_m)S_{\lambda^T}(y_1,\cdots,y_n) = \prod_{i=1}^{m}\prod_{j=1}^{n}(1+x_iy_j)$$

的组合证明。这里的求和是对于所有可能的形状 λ 进行的，而 λ^T 表示转置的形状。这些恒等式首先是由 D. E. Littlewood 发现的，*Proc. London. Math. Soc.* (2) **40** (1936), 40～70, 定理 V]。

注意：例如，由此得出，连续的二项式系数的任何乘积 $\binom{a}{k}\binom{a+1}{k}\cdots\binom{a+l}{k}$ 可由 $\binom{k}{k}\binom{k+1}{k}\cdots\binom{k+l}{k}$ 所整除，因为这个比是 $\Delta(a+l,\cdots,a+1,a,k-1,\cdots,1,0)/\Delta(l,\cdots,1,0)$。$\Delta(l,\cdots,1,0)=(l-1)!\ \cdots 1!\ 0!$ 的值有时称为"超阶乘"。

34. 一个钩的长度也是从钩的左下单元(x,y)到它的右上单元(x',y')的任何弯曲的通路的长度。我们证明一个更强的结果：如果有一个长度为 $a+b$ 的钩，则有一个长度为 a 的钩或者长度为 b 的钩。考虑单元$(x,y)=(x_1,y_1),(x_2,y_2),\cdots,(x_{a+b},y_{a+b})=(x',y')$，它们紧靠这个形状的下部。如果 $x_{a+1}=x_a$，则单元(x_a,y_1)有长度为 a 的一个钩；否则(x_{a+1},y_{a+b})有一个长度为 b 的钩。[参考 *Japanese J. Math*, **17** (1940), 165～184, 411～423。中山正是在排列群的研究中，最先考虑钩的人，而且他差不多接近于发现定理 H]。

35. 当 q_{ij} 增加时,步骤 G3-G5 的执行恰好使 p 数组的 h_{ij} 个元素减 1,因为这个算法沿着从 $p_{n'_j j}$ 到 p_{in_i} 的一条弯曲的通路进行。这些步骤的下一个执行或者由 j 的一个更大的值开始,或者在高于或等于以前转弯处停止。因此 q 数组从左到右和由底向上被填满。为了颠倒这个过程,我们从右向左和由顶向下进行:

H1. [初始化] 对于 $1\leqslant j\leqslant n_i$ 和 $1\leqslant i\leqslant n'_1$,置 $p_{ij}\leftarrow 0$。然后置 $i\leftarrow 1$ 和 $j\leftarrow n_1$。

H2. [求非 0 的单元] 如果 $q_{ij}>0$,转到步骤 H3 继续。否则如果 $i<n'_j$,则 i 加 1 并重复这个步骤。否则如果 $j>1$,则 j 减 1,置 $i\leftarrow 1$,并重复这个步骤。否则停止(q 数组现在为 0)。

H3. [减少 q,为转弯做准备] q_{ij} 减 1,并置 $l\leftarrow i, k\leftarrow n_i$。

H4. [下移或左移] 如果 $l<n'_k$ 和 $p_{lk}>p_{(l+1)k}$,则 l 加 1 并返回 H4。否则如果 $k>j$,则 k 减 1 并返回 H4。否则返回 H2。▌

对于一个给定的 j 头一个转弯的通路通过增加 $p_{n'_j j}$ 结束。因为 $p_{-1j}\leqslant\cdots\leqslant p_{n'_j j}$,意味着 $p_{n'_j j}>0$。对于列 j 的每个随继的通路停留在以前的通路之下或等于以前的通路,所以它也在 $p_{n'_j j}$ 处结束。在这途中遇到的不相等表示这个算法对其它的进行反序[*J. Combinatorial Theory* **A21** (1976),216~221]。

36. (a)所述的 z^m 系数是 $m=\sum h_{ij}g_{ij}$ 的解的个数,所以我们可以应用上一道题的结果。(b)如果 $a_1,\cdots,a_k$ 是任何正整数,我们可以通过对 k 用归纳法证明

$$[z^m]1/(1-z)(1-z^{a_1})\cdots(1-z^{a_k})=\binom{m}{k}\Big/a_1\cdots a_k+O(m^{k-1})$$

由习题 5.1.1-15,对于固定的 n,m 分划成至多 n 个部分的个数因此是 $\binom{m}{n-1}/n!+O(m^{n-2})$。这也是 $m=p_1+\cdots+p_n$ 分划成不同部分 $p_1>\cdots>p_n>0$ 的渐近个数(参见习题 5.1.1-16)。所以当有一个给定的 n 个单元形状的 N 个图表时,颠倒的平面分划的个数渐近地是 $N\binom{m}{n-1}/n!\ +O(m^{n-2})$。由部分(a),这也是 $\binom{m}{n-1}/\Pi h_{ij}+O(m^{n-2})$。[*Studies in Applied Math.* **50** (1971),167~188,259~279。]

37. 在一个矩形中的平面分划等价于颠倒的平面分划,所以钩的长度告诉我们在一个 $r\times c$ 的矩形中的生成函数 $1/\Pi_{i=1}^{r}\Pi_{j=1}^{c}(1-z^{i+j-1})$。设定 $r,c\to\infty$ 产生漂亮的答案 $1/(1-z)(1-z^2)^2(1-z^3)^3\cdots$。[MacMahon 原来在*Philosophical Transactions* **A211** (1912),75~110,345~373 中的推导,极其复杂。头一个相当简单的证明是由 Leonard Carlitz 发现的,*Acta Arithmetica* **13** (1967),29~47。]

38. (a)当 $k=l=1$ 时,概率是 $1/n$;否则对 $k+l$ 用归纳法,它是

$$\frac{nP(I\setminus\{i_0\},J)+nP(I,J\setminus\{j_0\})}{nd_{i_0j_0}}=\frac{(d_{i_0b}+d_{aj_0})/(nd_{i_0b}\cdots d_{i_{k-1}b}\,d_{aj_0}\cdots d_{aj_{l-1}})}{d_{i_0b}+d_{aj0}}$$

(b)对所有 I 和 J 求和给出

$$n^{-1}(1+d_{1b}^{-1})\cdots(1+d_{(a-1)b}^{-1})(1+d_{a1}^{-1})\cdots(1+d_{a(b-1)}^{-1})$$

容易看出,它等于 $f(T\setminus\{a,b\})/f(T)$。

(c)对所有的角求和产生 1,因为每一条通路都在一个角落处结束。因此 $\sum f(T\setminus\{(a,b)\})=f(T)$,而且通过对 n 用归纳法证明了定理 H。而且,如果我们把 n 放在随机通路的角落单元处并在剩下的 $n-1$ 个单元上重复这一过程,我们以 $1/f(T)$ 的概率得到每个图表。[*Advances in Math.* **31** (1979),104～109。]

39.(a)$Q_{11},\cdots,Q_{1n}$将是$b_1\cdots b_n$,即原来的排列 $P_{11}\cdots P_{1n}$的反序表。(参见 5.1.1 节)。

(b)$Q_{11},\cdots,Q_{n1}$是习题 5.1.1-7 的负的反序表$(-C_1)\cdots(-C_n)$。

(c)这个条件显然为步骤 P3 所保持。

(d)$\begin{pmatrix}1&4\\2&3\end{pmatrix}\to\left(\begin{pmatrix}13\\24\end{pmatrix},\begin{pmatrix}0&-1\\0&0\end{pmatrix}\right)$;$\begin{pmatrix}4&3\\1&2\end{pmatrix}\to\left(\begin{pmatrix}1&2\\3&4\end{pmatrix},\begin{pmatrix}0&-1\\0&0\end{pmatrix}\right)$。这个例子表明,如果不注视数组 P 我们不能向后运行步骤 P3。

(e)

12	10	8	14	15	11
9	13	7	1		
6	4	5			
16	3				
2					

(f)下列算法是正确的,但并不那么明显。

Q1.[对(i,j)进行循环] 以字典次序对数组的所有单元(i,j)实施步骤 Q2 和 Q3。(即是在每行中从顶向下,和从左到右);然后停止。

Q2.[调整 Q] 通过下列规则求出"头一个候选者"(r,s)。然后对于 $j\leqslant k<s$,置 $Q_{i(k+1)}\leftarrow Q_{ik}-1$。

Q3.[在(i,j)处拆开 P] 置 $K\leftarrow P_{rs}$。然后执行下列操作直到$(r,s)=(i,j)$为止:如果 $P_{(r-1)s}>P_{r(s-1)}$,置 $P_{rs}\leftarrow P_{(r-1)s}$和$r\leftarrow r-1$;否则置 $P_{rs}\leftarrow P_{r(s-1)}$和 $s\leftarrow s-1$,最后置 $P_{ij}\leftarrow K$。 ▌

在步骤 Q2 中,当 $s\geqslant j$ 以及 $Q_{is}\leqslant 0$ 且 $r=i-Q_{is}$时,单元(r,s)是一个候选者。设 T 是提示的有向树。算法 Q 的基本不变量之一是每当(r,s)是在步骤 Q2 中的一个候选者时,在 T 中将有从(r,s)到(i,j)的一条通路。该通路的颠倒可以通过字母 D、Q 和 R 的一个序列来进行编码。其意义是,我们在(i,j)处开始,然后向下(D)或者向右(R)或者停止(Q)。头一个候选者是在字母表的次序下字典次序的头一个;直观地说,它就是具有"最左和最下的"通路的候选者。

例如,在部分(e)的例子中当$(i,j)=(1,1)$时的候选者是(3,1),(4,2),(2,3),(2,4)和(1,6)。它们分别的编码是 DDQ, DDDRQ, RDRQ, RDRRQ,以及 RRRRRQ;所以头一个是(4,2)。

算法 P 是在*Funkts. Analiz i Ego Priloz.* **26**,**3**(1992),80～82 中不加证明地指出的一个构造的稍微简化了的版本。其正确性证明是不平凡的;J.-C. Novelli,I. Pak 和 A.V.Stoyanovskii 在*Disc. Math. and Theoretieal Comp. Sci.* **1** (1997),53～67 上给出了一个证明。

40. H. Rost,*Zeitschrift fur Wahrscheinlich-keitstheorie und verwandte Gebiete* ,**58** (1981),41～53。

41.(由 R.W.Floyd 给出的解)一个删去-插入操作实质上仅仅移动 a_i。在这样的操作的一个序列中,未被移动的元素保持它们的相对次序。因此如果可以通过 k 个删去-插入来对 π 排序,则它有长度为 $n-k$ 的一个递增的子序列;而且反之亦然。因此,$\mathrm{dis}(\pi)=n-$(π 的最长的递增子序列的长度)$=n+$(定理 A 中行 1 的长度)。

M.L.Fredman 已经证明为计算这个长度所需要的极小比较次数是 $n\lg n-n\lg\lg n+O(n)$[*Discrete Math.* **11** (1975),29～35]。

42.构造一个多重图,对于 $0\leqslant k\leqslant n$,它有顶点 $\{0_R,1_L,1_R,\cdots,n_L,n_R,(n+1)_L\}$和边 k_R—$(k+1)_L$;还包括边 0_R—7_R,7_L—1_L,1_R—2_L,2_R—4_L,4_R—5_L,5_R—3_L,3_R—6_R,6_L—8_L,它们定义了裂开炽体的“粘合”。精确地说,两个边接触每个顶点,因此连接的成分是循环的:$(0_R\ 1_L\ 7_L\ 6_R\ 3_R\ 4_L\ 2_R\ 3_L\ 5_R\ 6_L\ 8_L\ 7_R)(1_R\ 2_L)(4_R\ 5_L)$。任何触发操作都使循环数改变 -1,0 或 $+1$。因此我们至少需要五个触发来达到八个循环(0_R1_L) $(1_R2_L)\cdots(7_R8_L)$。[J. Kececioglu 和 D. Sankoff,*Algorithmica* **13** (1995),180～210。]

头一步必须打破粘合 6_L—8_L,因为在线性安排中当我们打破有相同的自左到右的两个粘合时,我们就得不到新循环。在一个触发之后,这保持五个可能性,即 $g_7^Rg_6g_3^Rg_5^Rg_4^Rg_2^Rg_1^R$,$g_7^Rg_1g_2g_4g_5g_3g_6$,$g_7^Rg_1g_2g_6g_3^Rg_5^Rg_4^R$,$g_7^Rg_1g_2g_4g_5g_6g_3^R$ 以及 $g_6g_3^Rg_5^Rg_4^Rg_2^Rg_1^Rg_7$;再有四个触发就足以把除了第二个之外的所有安排排序。

顺便说,$g_1\cdots g_7$ 有 $2^7\cdot 7!\ =645120$ 个不同的可能的安排,从吸烟的次序来说其中有 179904 其距离是≤5 的。

[S. Hannenhalli 和 P. Pevzher,*JACM* **46** (1999),1～27 给出了通过颠倒顺序来找出对任何带符号的排列进行排序最好方法的一个有效算法,而且,由 Kaplam、Shamir 和 Tarjin,*SICOMP* **29** (1999),880～892 作了改进以在 $O(n^2)$的时间内运行]。

43.通过带符号的排列 $\bar{7}12453\bar{6}$ 来表示像 $g_7^Rg_1g_2g_4g_5g_3g_6^R$ 这样一个安排。如果有一个负的元素,比如说 $\bar{k}$ 存在而不是 $\overline{k-1}$,一个触发将建立 2 循环$((k-1)_Rk_L)$。类似地,如果 $\bar{k}$ 存在而不是$\overline{k+1}$,一个触发建立$(k_R(k+1)_L)$。而且如果所有这特殊类型的触发删去所有负的元素,则单个触发就建立两个 2 循环。如果不出现有负的元素而且这个排列未排序,某个触发将保持循环的个数。因此如果给定的排列有一个负元素,我们可以在$\leqslant n$ 个触发中排序,否则则在$\leqslant n+1$ 个触发中排

序。

当 n 是偶数时，排列 $n(n-1)\cdots1$ 要求 $n+1$ 个触发，因为在头一个触发之后它有一个循环。当 $n>3$ 是奇数时，通过类似的论证排列 $213n(n-1)\cdots4$ 要求 $n+1$ 个触发。

44.设 c_k 是在前面答案中的多重图中长度为 $2k$ 的循环的个数。可以找出 c_k 的平均值的上限如下：可能的 $2k$ 循环的总数是 $2^k(n+1)^{\underline{k}}/(2k)$，因为我们可以以 $(n+1)^{\underline{k}}$ 种方法从 $\{0_R-1_L,\cdots,n_R-(n+1)_L\}$ 中选择 k 个不同边的序列，并且以 2^k 种方法来对它们定方向；这计算每个循环 $2k$ 次，包括像 $(1_R2_L2_R3_L)$ 或 $(1_R2_L3_L2_R3_R4_L)$ 或 $(1_R2_L6_R7_L4_L3_R2_R3_L6_L5_R)$ 这样不可能情况。当 $k\leqslant n$ 时，每个可能的 $2k$ 循环恰好出现在 $2^{n-k}(n-k)!$ 个带符号的排列中。例如，考虑 $k=5,n=9$ 的情况以及循环 $(0_R1_L9_L8_R7_R8_L1_R2_L5_L4_R)$。这个循环在多重图中出现当且仅当带符号的排列以 $\bar{4}$ 开始，并且包含子串 $\bar{9}18\bar{7}$ 和 $\bar{2}5$ 或它们的颠倒；通过找出 $\{1,2,3,6\}$ 的所有带符号的排列，并且以 $\bar{9}18\bar{7}$ 来代替 1，以 $\bar{2}5$ 来代替 2，我们得到所有的解。因此 $\mathrm{E}c_k\leqslant1/(2k)2^k(n+1)^{\underline{k}}2^{n-k}(n-k)!\ /2^nn!\ =\frac{1}{2}(1/k+1/(n+1-k))$。由此得出，$\mathrm{E}c=\sum_{k=1}^{n}\mathrm{E}ck+\mathrm{E}c_{n+1}<H_n+1$。由于 $n+1-c$ 是触发数的下限，我们需要 $\geqslant n+1-\mathrm{E}c>n-$ 它们中的 H_n。

[这个证明使用了 V. Bafna 和 P. Pevzner, *SICOMP* **25** (1996), 272～289 的思想。他们研究了通过颠倒顺序来对不带符号的排列排序的更困难的问题。在该问题中，证实了可以写成为不相交的循环的乘积(123)(345)(567)…，而且依赖于 n 是偶数还是奇数，而以 $(n-1\ n)$ 或 $(n-2\ n-1\ n)$ 结尾，是最难排序的。]

5.2 节

1.是；i 和 j 可以以任意顺序跑遍 $1\leqslant j<i\leqslant N$ 的值的集合，可能并行和/或作为记录被读入。

2.在这一章开始时给出的定义的意义下，这个排序是稳定的，因为这个算法实质上是对不同键码对 $(K_1,1),(K_2,2),\cdots,(K_N,N)$ 按字典顺序进行排序的。(如果我们把每个键码想像作通过它在文件中的位置向右边扩充，不出现相等的键码，因而这个排序是稳定的)。

3.它能排序，但不是以一种稳定的方式；如果 $K_j=K_i$ 且 $j<i$，则 R_j 将在排好了的顺序中出现在 R_i 之后，这个变化也将使程序 C 更慢地运行。

```
4.ENT1   N            1
  LD2    COUNT,1      N
  LDA    INPUT,1      N
  STA    OUTPUT+1,2   N
  DEC1   1            N
  J1P    *-4          N    ▌
```

5. 运行时间减少 $A+1-N-B$ 个单位，而且这在绝大多数情况下是个改进。

6. $u=0, v=9$。

在 D1 之后，COUNT = 0 0 0 0 0 0 0 0 0 0

在 D2 之后，COUNT = 2 2 1 0 1 3 3 2 1 1

在 D4 之后，COUNT = 2 4 5 5 6 9 12 14 15 16

在 D5 期间，COUNT = 2 3 5 5 5 8 9 12 15 16   j = 8

OUTPUT = … … … 1G … 4A … … 5L 6A 6T 6I 7O 7N … …

在 D5 之后，OUTPUT = 0C 0O 1N 1G 2R 4A 5T 5U 5L 6A 6T 6I 7O 7N 8S 9

7. 是，注意在步骤 D6 中 COUNT[K_j]被减值，而且 j 减值。

8. 它能排序，但不是以一种稳定的方式(参见习题 7)。

9. 设 $M=v-u$；假定 $|u|$，$|v|$ 能装入两个字节中。LOC(R_j)≡INPUT + j；LOC(COUNT[j])≡LOUNT + j；LOC(S_j)≡OUTPUT + j；rI1≡i；rI2≡j；rI3≡$i-v$ 或 rI3≡K_j。

```
M       EQU     V-U
KEY     EQU     0:2                      (在字节 3:5 中的随从信息)
1H      ENN3    M                  1     D1. 清除 COUNT
        STZ     COUNT+V,3        M+1     COUNT[v-k]←0
        INC3    1                M+1
        J3NP    *-2              M+1     u≤i≤v
2H      ENT2    N                  1     D2. 对 j 进行循环
3H      LD3     INPUT,2(KEY)       N     D3. COUNT[K_j]增值
        LDA     COUNT,3            N
        INCA    1                  N
        STA     COUNT,3            N
        DEC2    1                  N
        J2P     3B                 N     N≥j>0
        ENN3    M-1                1     D4. 累计
        LDA     COUNT+U            1     rA←COUNT[i-1]
4H      ADD     COUNT+V,3          M     COUNT[i-1]+COUNT[i]
        STA     COUNT+V,3          M     →COUNT[i]
        INC3    1                  M
        J3NP    4B                 M     u≤i≤v
5H      ENT2    N                  1     D5. 对 j 进行循环
6H      LD3     INPUT,2(KEY)       N     D6. 输出 R_j
        LD1     COUNT,3            N     i←COUNT[K_j]
        LDA     INPUT,2            N     rA←R_j
```

```
      STA     OUTPUT,1           N      S_i←rA
      DEC1    1                  N
      ST1     COUNT,3            N      COUNT[K_j]←i-1
      DEC2    1                  N
      J2P     6B                 N      N≥j>0  ▌
```

运行时间是$(10M+22N+10)u$。

10.为了避免使用 N 个额外"标志"位[见 1.3.3 小节和*Cybernetics* **1**(1965),95],并保持运行时间实质上同 N 成比例,我们可以使用下列以排列的循环结构为基础的算法:

P1.[对 i 进行循环] 对于 $1\leqslant i\leqslant N$ 做步骤 P2;然后结束这个算法。

P2.[$p(i)=i$ 吗?] 如果 $p(i)\neq i$,则进行步骤 P3 到 P5。

P3.[开始循环] 置 $t\leftarrow R_i, j\leftarrow i$。

P4.[修正 R_j] 置 $k\leftarrow p(j), R_j\leftarrow R_k, p(j)\leftarrow j, j\leftarrow k$。如果 $p(j)\neq i$ 则重复本步骤。

P5.[结束循环] 置 $R_j\leftarrow t, p(j)\leftarrow j$。▌

这个算法改变 $p(i)$,因为排序的应用允许我们假定 $p(i)$存储在内存中。另一方面,存在像矩阵转置这样一些应用,其中 $p(i)$是有待计算(为了节省内存空间而不造表)的 i 的函数。在这样一种情况下,我们可以使用下列的方法,对于 $1\leqslant i\leqslant N$ 实施步骤 B1 到 B3:

B1.置 $k\leftarrow p(i)$。

B2.如果 $k>i$,则置 $k\leftarrow p(k)$并重复这个步骤。

B3.如果 $k<i$,则什么也不做;但如果 $k=i$(这意味着 i 是在它的循环中最小的)则我们把包含 i 的循环排列如下:置 $t\leftarrow R_i$,然后当 $p(k)\neq i$ 时,重复地置 $R_k\leftarrow R_p(k)$,以及 $k\leftarrow p(k)$;最后 $R_k\leftarrow t$。▌

这个算法类似于 J. Boothroyd[*Comp. J.* **10** (1967),310]的过程,但它要求较少的数据移动;(I. D. G. Macleod)[*Australian Comp. J.* **2** (1970),16~19。]已经提出某些改进。对于随机排列,习题 1.3.3-14 中的分析表明,步骤 $B2$ 平均要实施$(N+1)H_N-N$ 步。请见习题 1.3.3-12 的答案中的参考文献。例如,如果习题 4 中的重新安排是通过 OUTPUT=INPUT 来完成的话,则类似的算法可被设计成以$(R_1,\cdots,R_N)$来代替$(R_{p(1)},\cdots,R_{p(N)})$。

11.命 $rI1\equiv i; rI2\equiv j; rI3\equiv k; rX\equiv t$。

```
1H    ENT1    N          1        P1.对i进行循环
2H    CMP1    P,1        N        P2. p(i)=i 吗?
      JE      8F         N        如果 p(i)=i 则转移
3H    LDX     INPUT,1    A-B      P3.开始循环,t←R_i
```

	ENT2	0,1	$A-B$	$j \leftarrow i$
4H	LD3	P,2	$N-A$	P4. 修正 R_j。$k \leftarrow p(j)$
	LDA	INPUT,3	$N-A$	
	STA	INPUT,2	$N-A$	$R_j \leftarrow R_k$
	ST2	P,2	$N-A$	$p(j) \leftarrow j$
	ENT2	0,3	$N-A$	$j \leftarrow k$
	CMP1	P,2	$N-A$	
	JNE	4B	$N-A$	如果 $p(j) \neq i$ 则重复
5H	STX	INPUT,2	$A-B$	P5. 结束循环。$R_j \leftarrow t$
	ST2	P,2	$A-B$	$p(j) \leftarrow j$
8H	DEC1	1	N	
	J1P	2B	N	$N \geq i \geq 1$ ▌

运行时间是$(17N-5A-7B+1)u$,这里 A 是排列 $p(1)\cdots p(N)$中的循环个数,且 B 是不动点(1-循环)的个数。由等式 1.3.3-(21)和 1.3.3-(28,)对于 $N\geq 2$,我们有 $A=(\min 1,\text{ave } H_N,\max N,\text{dev }\sqrt{H_N-H_N^{(2)}})$;$B=(\min 0,\text{ave } 1,\max N,\text{dev } 1)$。

12. 明显的方法是跑遍这个表列,并以数 k 代替第k 个元素的链接,然后在第二遍扫描中重新安排诸元素。下列更直接的方法是由 M.D.MacLaren 给出的,如果诸记录不太长,它更短和更快。(为方便起见假定对于 $1\leq$P$\leq N$,$0\leq$LINK(P)$\leq N$,这里 $\Lambda\equiv 0$)。

M1.[初始化] 置 P←HEAD,$k\leftarrow 1$。

M2.[完成了?] 如果 P$=\Lambda$(或者等价地,如果 $k=N+1$),算法结束。

M3.[确保 P$\geq k$] 如果 P$<k$,置 P←LINK(P),并重复这一步骤。

M4.[交换] 交换 R_k 和 R[P](假定 LINK(k)和 LINK(P)在这个过程中也交换),然后置 Q←LINK(k),LINK(k)←P,P←Q,$k\leftarrow k+1$,并返回步骤 M2。▌

MacLaren 方法有效的一个证明,可以以下列性质的归纳性验证为基础,这个性质在步骤 M2 开始时成立:在序列 P,LINK(P),LINK(LINK(P)),…,Λ 中$\geq k$ 的项是 $a_1,a_2,\cdots,a_{N+1-k}$,这里 $R_1\leq\cdots\leq R_{k-1}\leq R_{a_1}\leq\cdots\leq R_{a_{N+1-k}}$是这些记录所应有的最后顺序。而且对于 $1\leq j<k$ 有 LINK(j)$\geq j$,使得 LINK(j)$=\Lambda$ 意味着 $j\geq k$。

分析 MacLaren 算法是十分有趣的;它的突出性质之一是它可反向运行,即从 LINK(1),…,LINK(N)最后的值重新构造链接的原来集合。满足 $j\leq$LINK(j)$\leq N$ 的 $N!$ 种可能的输出配置的每一种,恰巧对应于 $N!$ 种可能的输入配置之一。如果 A 是步骤 M3 中 P←KINK(P)的次数,则 $N-A$ 是在这算法结束处使得 LINK(j)$=j$ 的j 的个数;这种情况当且仅当 j 是它循环中最大者时才出现;因此 $N-A$ 是排列中的循环个数,而且 $A=(\min 0,\text{ave } N-H_N,\max N-1)$。

参考文献:M.D.MacLaren,*JACM* **13** (1966),404~411;D.Gries 和 J.F.Prins, *Science of Computer Programming* **8** (1987),139~145。

13. **D5′**. 置 $r \leftarrow N$。

D6′. 如果 $r=0$，则停止。否则如果 COUNT[K_r]$<r$，则置 $r \leftarrow r-1$ 并重复这一步骤；如果 COUNT[K_r]$=r$，则 COUNT[K_r]和 r 都减 1，并重复这一步骤。否则置 $R \leftarrow R_r$，$j \leftarrow$COUNT[K_r]，COUNT[K_r]$\leftarrow j-1$。

D7′置 $S \leftarrow R_j$，$k \leftarrow$COUNT[K_j]，COUNT[K_j]$\leftarrow k-1$，$R_j \leftarrow R$，$R \leftarrow S$，$j \leftarrow k$。然后如果 $j \neq r$，则重复这一步骤；如果 $j=r$，则置 $R_j \leftarrow R$，$r \leftarrow r-1$，并转回到 D′6。 ▌

为了证明这个过程是正确的，注意在步骤 D′6 开始处，所有满足 $j>r$ 的不在它们最后的安放位置处的记录 R_j 都必须左移；当 $r=0$ 时，不可能有任何这样的记录，因为某个东西必须右移。这个算法是漂亮的，但对于相等的键码不稳定；它同定理 5.1.2B 中的 Foata 构造密切相关。

5.2.1 小节

1. 是。相等的元素决不彼此交错地移动。

2. 是。但当出现相等的元素时，运行时间将较慢，而排序将不稳定。

3. 下列 8 行被猜测为最短的 MIX 排序程序，尽管就速度来说，它是不值得推荐的。我们假定数出现在位置 $1,\cdots,N$ 上(即，INPUT EQU 0；否则需要多一行代码)。

```
2H      LDA    0,1    B
        CMPA   1,1    B
        JLE    1F     B
        MOVE   1,1    A
        STA    0,1    A
START   ENT1   N      A+1
1H      DEC1   1      B+1
        J1P    2B     B+1    ▌
```

注意：为了估计这个程序的运行时间，注意 A 是反序的个数。量 B 是反序表的相当简单的函数，而且(假定在随机顺序下的不同输入)它的生成函数是

$$z^{N-1}(1+z)(1+z^2+z^{2+1})\times$$
$$(1+z^3+z^{3+2}+z^{3+2+1})\cdots(1+z^{N-1}+z^{2N-3}+\cdots+z^{N(N-1)/2})/N!$$

B 的平均值是 $N-1+\sum_{k=1}^{N}(k-1)(2k-1)/6=(N-1)(4N^2+N+36)/36$；因此这个程序的平均运行时间大约是 $\frac{7}{9}N^3u$。

4. 在习题 5.1.1-7 的意义下，考虑给定的输入排列的反序表 $B_1\cdots B_N$。A 比等于 $j-1$ 的 B_j 的个数少 1，而 B 是诸 B_j 之和。因此当输入的排列是 $N\cdots 2\ 1$ 时，$B-A$ 和 B 都取极大值；当输入 $1\ 2\cdots N$ 时，两者取极小值，因此当 $A=0, B=0$ 时可达到时间的下限，即 $(10N-9)u$；当 $A=N-1$，$B=\binom{N}{2}$ 时出现极大值，这就是

$(4.5N^2+2.5N-6)u$。

5.生成函数是 z^{10N-9}乘以 $9B-3A$ 的生成函数。通过像在上题中那样考虑反序表,并回想起反序表中各项是彼此独立的,因此所求的生成函数是 $z^{10N-9}\prod_{1\leqslant j\leqslant N}((1+z^9+\cdots+z^{9j-18}+z^{9j-12})/j)$。方差成为 $2.25N^3+3.375N^2-32.625N+36H_N-9H_N^{(2)}$。

6.把输入区域当作一个循环表,而且位置 N 同位置 1 相邻。根据上一次插入的元素是落到已排序元素中心的右边还是左边,而分别地从当前的未排序元素段的左边或右边来取新的待插入的元素。过后,通常将需要"转动"这个区域,对于某个固定的 k,把每个记录顺着这个循环移动 k 个位置;这可像在习题 1.3.3-34 中一样有效地完成。

7.$|a_j-j|$的平均值是

$$\frac{1}{n}(|1-j|+|2-j|+\cdots+|n-j|)=\frac{1}{n}\left(\binom{j}{2}+\binom{n-j+1}{2}\right)$$

对 j 求和给出

$$\frac{1}{n}\left(\binom{n+1}{3}+\binom{n+1}{3}\right)=\frac{1}{3}(n^2-1)$$

顺便提及,所述和的方差可证明等于$[n>1](2n^2+7)(n+1)/45$。

8.否;例如,考虑键码 2 1 1 1 1 1 1 1 1 1 1。

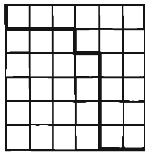
9.对于表 3,$A=3+0+2+1=6$,$B=3+1+4+21=29$;在表 4 中,$A=4+2+2+0=8$,$B=4+3+8+10=25$;因此程序 D 的运行时间分别为 $786u$ 和 $734u$。尽管移动的次数已经从 41 减少到 25,但运行的时间不能同程序 S 相匹敌,因为当 $N=16$ 时,有四次扫描的簿记时间是浪费掉的。当对 16 个项目排序时仅用两次扫描效果更好;一个两次扫描的程序 D 大约在 $N=13$ 时开始胜过程序 S,尽管在短时间内它们是几乎等同的(而对这样小的 N,程序长度也许是重要的)。

10.把"INC1 INPUT;ST1 0F(0:2)"插入行 07 和行 08 之间,并把行 10～17 改成

```
0H   CMPA   INPUT+N-H,1      NT-S
     JGE    7F               NT-S
3H   ENT2   N-H,1            NT-S-C
4H   LDX    INPUT,2          B
5H   STX    INPUT+H,2        B
     DEC2   0,4              B
     J2NP   6F               B
     CMPA   INPUT,2          B-A
     JL     4B               B-A
6H   STA    INPUT+H,2        NT-S-C  ▌
```

这里纯增了 4 条指令,但却节省了 $3(C-T)$个时间单位,这里 C 是 $K_j\geqslant K_{j-h}$的次数。在表 3 和表 4 中,节省的时间分别近似于 87 和 88;经验表明,$C/(NT-S)$的值

当 $h_{s+1}/h_s \approx 2$ 时大约是 0.4，而当 $h_{s+1}/h_s \approx 3$ 时大约是 0.3，所以这项改进是值得的。(另一方面，对于程序 S 作类似的改变是不合要求的，因为在该情况下的节省仅仅同 $\log N$ 成比例，除非已知输入是相当好地排了序的)。

11. 

12. 把└─ 变成为─┐总是使反序数改变 ±1，依赖于这个变动是在对角线上面还是下面而异。

13. 把权 $|i-j|$ 放到从 $(i, j-1)$ 到 (i, j) 的线段上。

14. (a)在 A_{2n} 的和式中交换 i 和 j，并把这两个和式加起来。(b)取这个结果的一半，我们看到

$$A_{2n} = \sum_{0 \leqslant i \leqslant j} (j-i) \binom{i+j}{i} \binom{2n-i-j}{n-j} = \sum_{i,k \geqslant 0} k \binom{2i+k}{i} \binom{2n-2i-k}{n-i-k}$$

因此 $\sum A_{2n} z^n = \sum_{k \geqslant 0} k z^k \alpha^{2k} / (1-4z) = z/(1-4z)^2$，其中 $\alpha = \left(1-\sqrt{1-4z}\right)/2z$。

以上证明是由 Leonard Carlitz 向作者提出的。另一个证明可以根据水平和垂直权之间的相互作用进行(参见习题 13)；利用习题 5.2.2-16 的答案中的恒等式，并令 $f(k)=k$，可得另一个证明；但是却未发现有关公式 $A_n = \lfloor n/2 \rfloor 2^{n-2}$ 的简单的组合推导。

15. 对于 $n>0$

$$\hat{g}_n(z) = z^n g_{n-1}(z); \quad \hat{h}_n(z) = \hat{g}_n(z) + z^{-n} \hat{g}_n(z)$$

$$g_n(z) = \sum_{k=1}^{n} \hat{g}_k(z) g_{n-k}(z); \quad h_n(z) = \sum_{k=1}^{n} \hat{h}_k(z) h_{n-k}(z)$$

命 $G(w,z) = \sum_n g_n(z) w^n$，我们发现 $wzG(w,z)G(wz,z) = G(w,z) - 1$。从这个表示我们可以导出，如果 $t = \sqrt{1-4w} = 1 - 2w - 2w^2 - 4w^3 - \cdots$，则我们有 $G(w,1) = (1-t)/(2w)$；$G_{\prime}(w,1) = 1/(wt) - (1-t)/(2w^2)$；$G'(w,1) = 1/(2t^2) - 1/(2t)$；$G_{\prime\prime}(w,1) = 2/(wt^3) - 2/(w^2 t) + (1-t)/w^3$；$G'_{\prime}(w,1) = 2/t^4 - 1/t^3$；且 $G''(w,1) = 1/t^3 - (1-2w)/t^4 + 10w^2/t^5$。此处下边的撇号表示相对于头一个参数的微商，而上边的撇号表示相对于第二参数的微商。类似地，由公式

$$w(zG(wz,z) + G(w,z))H(w,z) = H(w,z) - 1$$

我们导出

$$H'(w,1) = w/t^4, \quad H''(w,1) = -w/t^3 - w/t^4 + 2w/t^5 + (2w^2 + 20w^3)/t^7$$

这里概述的公式的处理原来是用手算的，但今天它可以很容易地由计算机来算。原则上，所有分布阶段都可以以这样的方式得到。

生成函数 $g_n(z)$ 也表示了 $\sum z^{\text{内部路径长度}}$，这里求和对具有 $n+1$ 个节点的所有树进行；参见习题 2.3.4.5-5。说明这样一点是有趣的，即 $G(w,z)$ 等于 $F(-wz,z)/F(-w,z)$ 其中 $F(z,q)=\sum_{n\geq 0}z^n q^{n^2}/\prod_{k=1}^{n}(1-q^k)$；在 $F(z,q)$ 中 $q^m z^n$ 的系数是把 $m=p_1+\cdots+p_n$ 如下分划的个数，即使得对于 $1\leq j<n$，$p_j\geq p_{j+1}+2$，且 $p_n>0$（参见习题 5.1.1-16）。

16. 当 $h=2$ 时，显然地对于通过格子图式右上角的通路出现极大值，即是

$$\binom{\lfloor n/2\rfloor+1}{2}$$

对于一般的 h，对应的数是

$$\hat{f}(n,h)=\binom{h}{2}\binom{q+1}{2}+\binom{r}{2}(q+1)$$

这里的 q 和 r 在定理 H 中定义：因为具有 $a_{i+jh}=1+q(h-i)+(r-i)[i\leq r]$ 对于 $1\leq i\leq h$ 和 $j\geq 0$ 的排列，使 $\binom{h}{2}$ 对排好序的子序列中的每一对之间的反序个数极大化。如果我们在(6)中以 $\hat{f}$ 代替 f，便得到极大的移动次数。

17. 有 $\binom{n+1}{2}$ 个反序的 $\{1,2,\cdots,2n\}$ 的惟一的二有序排列是 $n+1\ 1\ n+2\ 2\cdots 2n\ n$。递归地利用这一思想，我们得到通过对序列 $(2^t-1)^R\cdots 1^R 0^R$ 的每个元素加 1 来定义的排列，这里 R 表示把一个整数写成为一个 t 位二进数的操作，然后逆转这些数字从左到右的顺序！

18. 取出一个公共因子并命 $h_t=4N/\pi$；当 $h_0=1$ 时我们要把 $\sum_{s=1}^{t}h_s^{1/2}/h_{s-1}$ 极小化。微商之得到 $h_s^3=4h_{s-1}^2 h_{s+1}$，而且我们求得 $(2^t-1)\lg h_1=2^{t+1}-2(t+1)+\lg h_t$。所述的估计的极小值成为 $(1-2^{-t})\pi^{(2^{t-1}-1)/(2^t-1)}N^{1+2^t-1/(2^t-1)}/2^{1+(t-1)/(2^t-1)}$。当 $t\to\infty$ 时它迅速地趋于 $N\sqrt{\pi N}/2$。

当 $N=1000$ 时"最优的"h 的典型例子是(也看表 6)：

$$h_2\approx 57.64,\quad h_1\approx 6.13,\quad h_0=1;$$

$$h_3\approx 135.30,\quad h_2\approx 22.05,\quad h_1\approx 4.45,\quad h_0=1;$$

$$h_4\approx 284.46,\quad h_3\approx 67.23,\quad h_2\approx 16.34,\quad h_1\approx 4.03,\quad h_0=1;$$

$$h_9\approx 9164.74,\quad h_8\approx 12294.05,\quad h_7\approx 7119.55,\quad h_6\approx 2708.95,\quad h_5\approx 835.50,$$

$$h_4\approx 232.00,\quad h_3\approx 61.13,\quad h_2\approx 15.69,\quad h_1\approx 3.97,\quad h_0=1。$$

19. 命 $g(n,h)=H_r-1+\sum_{r<j\leq h}q/(qj+r)$，这里 q 和 r 在定理 H 中定义；然后在(6)中以 g 代替 f。

20.（把它写出来比理解它更为困难。）假设一个 k 有序文件 $R_1,\cdots,R_N$ 已经进行了 h 排序，并设 $1\leq i\leq N-k$；我们要来证明 $K_i\leq K_{i+k}$。求 u,v 使得 $1\leq u,v\leq h$，$i\equiv u$，且 $i+k\equiv v$ (modulo h)；并对于 $x_j=K_{v+(j-1)h}$，$y_j=K_{u+(j-1)h}$ 应用定理

L。于是诸 y 的头 r 个元素 $K_u, K_{u+h}, \cdots, K_{u+(r-1)h}$ 分别≤诸 x 的最后 r 个元素 $K_{u+k}, K_{u+k+h}, \cdots, K_{u+k+(r-1)h}$，其中 r 是使得 $u+k+(r-1)h \leq N$ 的最大整数。

21. 如果 $xh+yk=x'h+y'k$，我们有 $(x-x')h=(y'-y)k$，所以对于某个整数 t，有 $x'=x+tk$ 和 $y'=y-th$。令 $h'h+k'k=1$；于是 $n=(nh')h+(nk')k$，所以每一个整数 n 有形如 $n=xh+yk$ 的惟一表示，其中 $0\leq x<k$，且 n 是可生成的当且仅当 $y\geq 0$。类似地，令 $hk-h-k-n=x'h+y'k$；于是 $(x+x')h+(y+y')k=hk-h-k$，因此 $x+x'\equiv k-1 \pmod{k}$，因而我们必定有 $x+x'=k-1$。因此 $y+y'=-1$ 和 $y\geq 0$ 当且仅当 $y'<0$。

这个结果的对称性表明，在所述的形式下，恰有 $\frac{1}{2}(h-1)(k-1)$ 个正整数是不可表示的，这是原来由 Sylvester 提出的［*Mathematical Questions, with their Solutions, from the "Educational Times"*, **41** (1884), 21］。

22. 为了避免麻烦的记号，考虑 $s=4$，这代表了一般情况。设 n_k 是同余于 k (modulo 15)而且可以以 $15a_0+31a_1+\cdots$ 的形式表示的最小数；于是我们很容易求得

k =	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
n_k =	0	31	62	63	94	125	126	127	158	189	190	221	252	253	254

因此 $239=2^4(2^4-1)-1$ 是最大的不可表示的数，而且不可表示的数的总数是

$$x_4=(n_1-1+n_2-2+\cdots+n_{14}-14)/15=(2+4+4+6+8+8)+8+(10+12+12+14+16+16)+16=2x_3+8\cdot 9$$

一般，$x_s=2x_{s-1}+2^{s-1}(2^{s-1}+1)$。

对于另一个问题，答案分别是 $2^{2s}+2^s+2$ 和 $2^{s-1}(2^s+s-1)+2$。

23. N 个数的每一个在它的子文件中至多有 $\lfloor (h_{s+2}-1)(h_{s+1}-1)/h_s \rfloor$ 个反序。

24. (同 V. Pratt 一起得到的解。) 构造 $\{1,2,\cdots,N\}$ 的"h 违例排列"如下。以 $a_1 \cdots a_N$ 个空白开始；然后对于 $j=2,3,4,\cdots$ 执行步骤 j：每当 $(2^h-1)j-i$ 是如同习题 22 中那样可表示的一个正整数时，用在这个排列中还未出现的最小数从左到右地填入诸空白位置 a_i。继续进行，直到所有位置都填满为止。于是对于 $N=20$ 这个 2 违例排列是

6 2 1 9 4 3 12 7 5 15 10 8 17 13 11 19 16 14 20 18

对于所有的 $k\geq h$ 的 h 违例排列是 (2^k-1) 有序的。当 $2^h<j\leq N/(2^h-1)$ 时，在步骤 j 期间恰好填满 2^h-1 个位置。它们中的第 $k+1$ 个至少增加 $2^{h-1}-2k$ 到移动次数上。这些移动是对排列进行 $2^{h-1}-1$ 排序所要求的。因此当 $N=2^{h+1}(2^h-1)$ 时为排序具有增量 $h_s=2^s-1$ 的 h 违例排列的移动次数是 $>2^{3h-4}>\frac{1}{64}N^{3/2}$。Pratt 在他的博士论文(斯坦福大学，1972)中，把这个构造推广到很大的一类相似序列，包括(12)在内。H. Erkiö BIT **20**(1980)，130～136 讨论了需要甚至更多移动的寻找排列的带启发式探索。关于对 Pratt 构造的改进，也请见 Weiss 和 Sedgewick, *J. Algo-*

rithms ,**11** (1990),242～251。

25. F_{n+1}。[这一结果是由 H.B.Mann 得出的,*Econometrica* **13** (1945),256];因为这个排列必须由 1 或 2 1 开头,至多有$\lfloor n/2 \rfloor$个反序;而且反序的总数是

$$\frac{N-1}{5}F_N+\frac{2N}{5}F_{N-1}$$

(见习题 1.2.8-12)。注意,F_{N+1}个排列可以方便地由点和破折号的"Morse 代码"序列表示,这里破折号对应于反序;参见习题 4.5.3-32。因此我们已经发现在长度为 N 的所有 Morse 代码序列中破折号的数目。

我们的推导表明,一个随机的 3 和 2 有序排列大约有$\frac{1}{5}(\phi^{-1}+2\phi^{-2})N=\phi^{-1}N/\sqrt{5}\approx.276N$ 个反序。但如果一个随机排列是 3 有序的,则它也是 2 有序的,习题 42 表明,它有$\approx N/4$ 个反序;如果它是 2 有序的,则它也是 3 有序的,它有$\approx N/3$。

26. 是的;一个最短的例子是 4　1　3　7　2　6　8　5,它有 9 个反序。一般地说,对于 $-1\leqslant s\leqslant 1$,构造 $a_{3k+s}=3k+4s$ 产生 3-有序,5-有序,7-有序的一些文件,并有近似于$\frac{4}{3}N$ 个反序。当 $N \bmod 3=2$ 时,这个构造是最好的。

27. (a)参见*J. Algorithms* **15** (1993),101～124。C.G.Plaxton 和 T.Suel,*J. Algorithms* **23** (1997),221～240 独立地发现了一个更简单的证明。它表明,c 可以是任何$<1/2$ 的常数。(b)如果 $m>\frac{1}{4}c^2(\ln N/\ln\ln N)^2$ 这是显然的。否则 $N^{1+c/\sqrt{m}}\geqslant N(\ln N)^2$。R.E Cypher[*SICOMP* **22** (1993),62～71]已经证明当对所有的 s,增量满足 $h_{s+1}>h_s$ 和当一个排序网络像在习题 5.3.4-2 中那样构造时,稍微更强的上限是 $\Omega(N(\log N)^2/\log\log N)$。对于渐近的平均运行时间,还不知道非平凡的下限。

28. 由(11),209 109 41 19 5 1,但可能还有更好的序列;见习题 29。

29. 1971 年 C. Tribolet 的实验,得到选择值 373 137 53 19 7 3 1($B_{\text{ave}}\approx 7210$)及 317 101 31 11 3 1($B_{\text{ave}}=8170$)。[这些值中的头一个需要$\approx 127720\,u$ 的排序时间,相对照的是当利用增量(11)对相同的数据进行排序时为$\approx 128593u$]。一般说来,Tribolet 建议命 h_s 是最接近于$N^{s/t}$的质数。1972 年 Shelby Siegel 的实验指出,对于 $N\leqslant 10000$,在这样的方法下最好的增量数是 $t\approx\frac{4}{3}\ln(N/5.75)$。

另一方面,Marcin Ciura 在 2001 年的实验指出,增量 229 96 41 19 10 4 1 可得到最小的 7 遍的 B_{ave}(≈ 6879),虽然序列 737 176 69 27 10 4 1 产生最小的总时间($\approx 125077u$)。

根据 Carole M. McNamee 所作的广泛测试,最好的 3 增量序列似乎是 45 7 1($B_{\text{ave}}\approx 18240$)。对于 4 增量,在她的测试中 91 23 7 1 是优胜者($B_{\text{ave}}\approx 11865$),但是一个颇为广泛的增量范围也大体给出相同的性能。

30. 在三角区域$\{x\ \ln 2+y\ \ln 3<\ln N,x\geqslant 0\quad y\geqslant 0\}$中的整数点的个数是$\frac{1}{2}$

$(\log_2 N)(\log_3 N)+O(\log N)$。由定理 K，在我们进行 h 排序时，这个文件已经是 $2h$ 有序的和 $3h$ 有序的；因此习题 25 在此适用。

31.

```
START  ENT3  T                  1
1H     LD4   H,3                T
       ENN2  -INPUT-N,4         T
       ST2   6F(0:2)            T
       ST2   7F(0:2)            T
       ST2   4F(0:2)            T
       ENT2  0,4                T
       JMP   9F                 T
2H     LDA   INPUT+N,1          NT-S-B+A
4H     CMPA  INPUT+N-H,1        NT-S-B+A
       JGE   8F                 NT-S-B+A
6H     LDX   INPUT+N-H,1        B
       STX   INPUT+N,1          B
7H     STA   INPUT+N-H,1        B
       INC1  0,4                B
8H     INC1  0,4                NT-B+A
       J1NP  2B                 NT-B+A
       DEC2  1                  S
9H     ENT1  -N,2               T+S
       J2P   8B                 T+S
       DEC3  1                  T
       J3P   1B                 T    ▌
```

在程序 D 中 A 是和自左至右的极大值有关的量，在这里和在那里一样 A 也是同自右至左的极大值有关的。这两个量都有相同的统计特性。在内循环中的简化已经把运行时间减少到 $7NT+7A-2S+1+15T$ 单位，奇怪，竟同 B 无关！

当 $N=8$ 时，增量是 6，4，3，2，1，而且我们有 $A_{\text{ave}}=3.892$，$B_{\text{ave}}=6.762$；平均总的运行时间是 $276.24u$（对照表 5）。A 和 B 两者在排列 7 3 8 4 5 1 6 2 中都取极大值。当 $N=1000$ 时，有 40 个增量，972，864，768，729，…，8，6，4，3，2，1；类似于表 6 中的经验测试给出 $A\approx 875$，$B\approx 4250$，而且总的时间大约是 $268000u$（比采用习题 28 增量的程序 D 的两倍还长）。

我们不在一个辅助表中存储增量，而是很方便地在一台二进机器上生成它们如下：

P1. 置 $m\leftarrow 2^{\lceil \lg N\rceil -1}$，小于 N 的 2 的最大幂。

P2. 置 $h\leftarrow m$。

P3. 使用 h 作为一次排序扫描的增量。

P4. 如果 h 是偶数，则置 $h \leftarrow h + h/2$；然后如果 $h < N$，则返回到 P3。

P5. 置 $m \leftarrow \lfloor m/2 \rfloor$，而且如果 $m \geqslant 1$ 则返回 P2。 ▌

尽管增量不以递减次序来生成，但这里确定的次序足以使排序算法是有效的。

32. 4 12 11 13 2 0 8 5 10 14 1 6 3 9 16 7 15。

33. 可以作出两种类型的改进。首先，假定人为的键码 K_0 是 ∞，这样我们可以不必测试 p 是否 >0。(例如，这个思想已经用于算法 2.2.4A 中。)其次，一个标准的“优化”技术：我们可以设置两份内循环，且把对 p 和 q 的寄存器赋值相交换；这就避免了 $q \leftarrow p$ 的赋值(这个思想已经用于习题 1.1-3 中)。

于是，我们假定单元 INPUT 在它的(0:3)字段中包含最大可能的值，而且以下列程序代替程序 L 中的行 07 和它后面的诸行：

```
07  8H  LD3   INPUT,2(LINK)   B′   p←L_q(这里 p≡rI3, q≡rI2)
08      CMPA  INPUT,3(KEY)    B′
09      JG    4F              B′   如果 K > K_P 则以 q↔r 转到 L4
10  7H  ST1   INPUT,2(LINK)   N′   L_q←j
11      ST3   INPUT,1(LINK)   N′   L_j←p
12      JMP   6F              N′   转到减少 j
13  4H  LD2   INPUT,3(LINK)   B″   p←Lq(这里 p≡rI2, q≡rI3)
14      CMPA  INPUT,2(KEY)    B″
15      JG    8B              B″   如果 K > K_P 则以 q↔P 转到 L4
16  5H  ST1   INPUT,3(LINK)   N″   L_q←j
17      ST2   INPUT,1(LINK)   N″   L_j←p
18  6H  DEC1  1               N    j←j−1
19      ENT3  0               N    q←0
20      LDA   INPUT,1         N    K←K_j
21      J1P   4B              N    N > j ≥ 1  ▌
```

这里 $B' + B'' = B + N - 1$，$N' + N'' = N - 1$，所以总共的运行时间为 $5B + 14N + N' - 3$ 单位。由于 N' 是在其右边有奇数个较小元素的元素个数，因此它有下列统计数字：

$$\left(\min 0, \text{ave } \frac{1}{2}N + \frac{1}{4}H_{\lfloor N/2 \rfloor} - \frac{1}{2}H_N, \max N - 1\right)$$

∞ 的技巧也可以加速程序 S；J. H. Halperin 建议的以下代码使用了这一思想以及 MOVE 指令，从而使运行时间减少到 $(6B + 11N - 10)u$，其中假定 INPUT + N + 1 单元中已经包含有最大可能的单字值：

```
01  START   ENT2   N-1           1
02  2H      LDA    INPUT,2       N-1
03          ENT1   INPUT,2       N-1
04          JMP    3F            N-1
05  4H      MOVE   1,1(1)        B
06  3H      CMPA   1,1           B+N-1
07          JG     4B            B+N-1
08  5H      STA    0,1           N-1
09          DEC2   1             N-1
10          J2P    2B            N-1     ■
```

加倍内循环还将另外节省 $B/2$ 左右的时间单位。

34.有 $\binom{N}{n}$ 个 N 种选择的序列,其中给定的表被选择 n 次;每个这样的序列出现的概率是 $(1/M)^n(1-1/M)^{N-n}$,因为给定的表以 $1/M$ 的概率被选择。

35.
```
24       ENT1  0                 1
25       ENT2  1-M               1
26  7H   LD3   HEAD+M,2          M
27       J3Z   8F                M
28       ST3   INPUT,1(LINK)     M-E
29       ENT1  0,3               N
30       LD3   INPUT,1(LINK)     N
31       J3P   *-2               N
32  8H   INC2  1                 M
33       J2NP  7B                M   ■
```

注意:如果通过在行 19 和 20 之间插入“STI END 4”来修改程序 M 以记住每个表列当前的末尾,我们可以像在算法 5.2.2H 那样把表列钩在一起而节省时间。

36.程序 L:$A=3$,$B=41$,$N=16$,时间 $=496u$。程序 M:$A=2+1+1+3=7$,$B=2+0+3+3=8$,$N=16$,时间 $=549u$。(我们还应该加上习题 35 需要的时间,$94u$,以便进行严格地公平的比较。乘法是很慢的!还注意习题 33 中改进的程序 L 只花费 $358u$。)

37.所述的恒等式等价于

$$g_{NM}(z) = M^{-N}\sum_{n_1+\cdots+n_M=N}\left(\frac{N!}{n1!\cdots n_M!}\right)g_{n_1}(z)\cdots g_{n_M}(z)$$

这可以像在习题 34 中那样来证明。把其中某些生成函数列成表,以指出 M 增长的趋势,可能是有趣的:

$g_{41}(z)=(216+648z+1080z^2+1296z^3+1080z^4+648z^5+216z^6)/5184$

$g_{42}(z)=(945+1917z+1485z^2+594z^3+135z^4+81z^5+27z^6)/5184$

$g_{43}(z)=(1704+2264z+840z^2+304z^3+40z^4+24z^5+8z^6)/5184$

如果 $G_M(w,z)$是所述双重生成函数，对 z 求微商给出

$$G'_M(w,z)=M\left(\sum_{n\geqslant 0}g_n(z)\frac{w^n}{n!}\right)^{M-1}\sum_{n\geqslant 0}g'_n(z)\frac{w^n}{n!}$$

因此

$$\sum_{N\geqslant 0}g'_{MN}(1)\frac{M^Nw^N}{N!}=Me^{(M-1)w}\left(\frac{w^2}{4}e^w\right)=\frac{M}{4}w^2e^{Mw};$$

类似地，公式 $g''_n(1)=\frac{3}{2}\binom{n}{4}+\frac{5}{3}\binom{n}{3}$产生

$$\sum_{N\geqslant 0}g''_{NM}(1)\frac{M^Nw^N}{N}=M(M-1)e^{(M-2)w}\left(\frac{w^2}{4}e^w\right)^2+Me^{(M-1)w}\left(\frac{w^4}{16}+\frac{5}{16}w^3\right)e^w$$

把 w^N 的系数等置给出 $g'_{NM}(1)=\frac{1}{2}\binom{N}{2}M^{-1}$，$g''_{NM}(1)=\left(\frac{3}{2}\binom{N}{4}+\frac{5}{3}\binom{N}{3}\right)M^{-2}$，因此方差是$\left(\frac{1}{6}\binom{N}{3}+\frac{2M-1}{4}\binom{N}{2}\right)M^{-2}$。

38. $\sum_{j,n}\binom{N}{n}p_j^n(1-p_j)^{N-n}\binom{n}{2}=\binom{N}{2}\sum_j p_j^2$；置 $p_j=F(j/M)-F((j-1)/M)$，且 $F'(x)=f(x)$，当 F 有相当好的特性时，这近似于$\binom{N}{2}/M$ 乘 $\int_0^1 f(x)^2dx$。[然而，$\int_0^1 f(x)^2dx$ 可能十分大。关于适用于所有有限的可积密度的一个改进，请见定理 5.2.2T]。

39. 为了极小化 $AC/M+BM$，我们需要 $M=\sqrt{AC/B}$，所以 M 是刚好高于或低于这个量的整数之一。(在程序 M 的情况下，我们将把 M 选成同N 成比)。

40. 通过把 k 限定为$O(N^{1+\epsilon})$，把$(1-\alpha/N)^k$ 展开为 $e^{-\alpha k/N}$乘以$(1-k\alpha^2/2N^2+\cdots)$，并且使用欧拉求和公式，可以得到对于

$$\sum_{n>N}n^{-1}(1-\alpha/N)^{n-N}=-N^{-1}+\sum_{k\geqslant 0}(N+k)^{-1}(1-\alpha/N)^k$$

的渐近级数。它以项 $e^\alpha E_1(\alpha)(1+\alpha^2/2N)-(1+\alpha)/2N+O(N^{-2})$开始。因此(15)的渐近值是 $N(\ln\alpha+\gamma+E_1(\alpha))/\alpha+(1-e^{-\alpha}(1+\alpha))/2\alpha+O(N^{-1})$。[对于$\alpha=1,2,10,$N 的系数分别≈0.7966，0.6596，0.2880。]注意，由习题 5.2.2-43，我们有 $\ln\alpha+\gamma+E_1(\alpha)=\int_0^\alpha(1-e^{-t})t^{-1}dt$。

41. (a)我们有 $a_k=O(\rho^k)$，因为质数定理意味着在 ρ^k 和 ρ^{k+1}之间的质数个数是$(\rho^{k+1}/(k+1)-\rho^k/k)/\ln\rho+O(\rho^k/k^2)$；对于所有充分大的 k 来说这为正。因此(10)的头$\binom{k}{2}$个元素之和是$\sum_{1\leqslant i<j\leqslant k}b(a_i,a_j)=\sum_{1\leqslant i<j\leqslant k}O(\rho^{i+j})$；而且我们有

$$\sum_{1\leqslant i<j\leqslant k}\rho^{i+j}=\frac{\rho^3(\rho^k-1)(\rho^{k-1}-1)}{(\rho^2-1)(\rho-1)}$$

(b) 如果 $\binom{k-1}{2} < \log_\rho N \leqslant \binom{N}{2}$，我们有 $(k-2)^2 < 2\log_\rho N$，因此 $\rho^{2k} = O(\exp\sqrt{\ln N})$。

注意当 $\rho \to 1$ 时，基序列 $a_1, a_2, \cdots$ 变成等于质数序列，而且定理 I 的上限简化为 $O(N(\log N)^4(\log\log N)^{-2})$。

42. (a) [姚期智，*J. Algorithms* **1** (1980)，14～50]。我们可以证明 $\binom{h}{2}$ 对表列的每一对贡献 $\frac{\sqrt{\pi}}{4}g^{-2}h^{-3/2}N^{3/2} + O(N/gh)$ 个反序给每个子文件 $(K_a, K_{a+g}, K_{a+2g}, \cdots)$，$1 \leqslant a \leqslant g$。例如，假设 $h=12$，$q=5$，$a=1$，并且考虑表列 $K_3 < K_{15} < K_{27} < \cdots$ 和 $K_7 < K_{19} < K_{31} < \cdots$ 同子文件 $(K_1, K_6, K_{11}, \cdots)$ 相交中的反序。在头一次扫描之后，$(K_3, K_7, K_{15}, K_{19}, K_{27}, K_{31}, \cdots)$ 是一个随机 2 有序的排列。我们所关心的元素 K_j 有 $j \equiv 1$ (modulo 5) 和 $j \equiv 3$ 或 7 (modulo 12)；因此 $j \equiv 51$ 或 31 (modulo 60)。因此我们要来计算 $g(51, 31)$ 的平均值。其中

$$g(x,y) = \sum_{j<k}([K_{x+ghj} > K_{y+ghk}] + [K_{y+ghj} > K_{x+ghk}]) + r(x,y)$$

$$r(x,y) = \sum_j [K_{\min(x,y)+ghj} > K_{\max(x,y)+ghj}] < N/gh + 1$$

如果 $|p| \leqslant g$ 和 $|q| \leqslant g$，我们有

$$[K_{j+ph-gh} > K_{k+qh+gh}] \leqslant [K_j > K_k] \leqslant [K_{j+ph+gh} > K_{k+qh-gh}];$$

因此

$$[K_{x+ghj} > K_{y+ghk}] + [K_{y+ghj} > K_{x+ghk}] \leqslant$$
$$[K_{x+ph+gh(j+1)} > K_{y+qh+gh(k-1)}] + [k_{y+qh+gh(j+1)} > K_{x+ph+gh(k-1)}]$$

而且由此得出 $g(x,y) \leqslant g(x+ph, y+qh) + 8N/gh$。类似地我们求得 $g(x,y) \geqslant g(x+ph, y+qh) - 8N/gh$。但对于任何给定的 $b \neq c$，对于使得 $x \bmod h = b$ 和 $y \bmod h = c$ 的所有 g^2 对 (x,y)，$g(x,y)$ 之和是在 $2N/h$ 个元素的一个随机 2 有序排列的反序的总数。因此由习题 14，$g(x,y)$ 的平均值为 $g^{-2}\sqrt{\pi/128}(2N/h)^{3/2} + O(N/gh)$。

(b) 参见 S. Janson 和 D. E. Knuth，*Random Structures and Algs.* **10** (1997)，125～142。当 h 和 g 很大时，我们有 $\psi(h,g) = \sqrt{\pi h/128}\,g + O(g^{-1/2}h^{1/2}) + O(gh^{-1/2})$。

43. 如果在步骤 D3 之后 $K < K_l$，则置 $(K_l, \cdots, K_{j-h}, K_j) \leftarrow (K, K_l, \cdots, K_{j-h})$；否则执行步骤 D4 和 D5 直到 $K \geqslant K_i$。这里当 $j = h+1$ 时 $l = 1$，当 j 增加 1 时，$l \leftarrow l + 1 - h[l = h]$。[参见 H. W. Thimbleby，*Software Practice & Exper.* **19** (1989)，303～307]。无论如何，使用适当的增量序列，内循环将不经常执行使得这个改进有必要。

加速这个程序的另一个想法 [参见 W. Dobosiewicz，*Inf. Proc. Letters*，**11** (1980)，5～6] 是当 $h > 1$ 时仅仅部分地排序，而不试图使 K_j 传播到比 $j-h$ 更左的

位置;但该方法似乎要求更多的增量。

44.(a)是的。每当 π' 是比 π 高一步时,这是显然的,而且习题 5.1.1-29 表明,有从 π 到它上面的任何排列的相邻转置的一条通路。

(*b*)是的。类似地,如果 π 在 π' 之上,则 π^R 是在 π'^R 之下。

(c)否;2 1 3 既不在 3 1 2 之上也不在它之下,但 2 1 3≤3 1 2。

[偏序 $\pi\leqslant\pi'$ 首先是由 C. Ehresmann,*Annals of Math*. (2) **35** (1934),396~443 §20 在代数拓扑的范畴中讨论的。许多数学家现在把它叫做排列的"Bruhat 次序",而在上性叫做"弱 Bruhat 次序"——尽管在上性实际上是一个更强的条件,因为它不太常成立。仅仅弱的次序定义一个格]。

5.2.2 小节

1. 否;它少 $2m+1$ 个反序,其中 $m\geqslant 0$ 是使得 $i<k<j$ 且 $a_i>a_k>a_j$ 的元素 a_k 的个数。(因此所有的交换排序方法最终都将收敛到一个排好序的排列。)

2. (a)6。(b) [A. Cayley,*Philosophical. Mag*. (3) **34** (1849),527~529.]考虑 π 的循环表示。在同一循环中的任何元素交换,都使循环数加 1;在不同循环中的任何元素交换都使循环数减 1。(这实际上是习题 2.2.4-3 的内容。)一个完全排好序的排列通过有 n 个循环来表征。因此 xch(π)是 n 减 π 中的循环个数。[算法 5.2.3S恰恰进行 xch(π)个交换;见习题 5.2.3-4。]

3. 是;相等的元素决不能彼此交叉地移动。

4. 它是在反序表中 $b_1>\max(b_2,\cdots,b_n)$ 的概率,即

$$\Big(\sum_{1\leqslant k<n}k!\,k^{n-k-1}\Big)\Big/n! = \sqrt{\pi/2n}+O(n^{-1}) = \text{可忽略的}$$

5. 我们可以假定 $r>0$。设 $b_i'=(b_i-r+1)[b_i\geqslant r]$是在 $r-1$ 次扫描之后的反序表。如果 $b_i'>0$,则元素 i 就居于b_i'个更大的元素之后,这些更大的元素中的最大者至少将往上冒到 b'_i+i 的位置,因为有 i 个元素$\leqslant i$。而且如果元素 j 是有待交换的最右者,则在第 r 次扫描之后,我们有 $b_j'>0$ 和 BOUND $=b_j'+j-1$。

6. 解法 1:一个被放置在离它最后位置右边最远的元素,在每次扫描中都左移一步,最后一次扫描除外。解法 2(更高级的):由习题 5.1.1-8,答案(f),对于 $1\leqslant i\leqslant n$,$a_i'-i=b_i-c_i$,其中 $c_1c_2\cdots c_n$ 是对偶的反序表。如果 $b_j=\max(b_1,\cdots,b_n)$则 $c_j=0$。

7. $(2(n+1)(1+P(n)-P(n+1))-P(n)-P(n)^2)^{1/2}=\sqrt{(2-\pi/2)n}+O(1)$。

8. 当 $i<k+2$ 时,对于 b_i 有 $j+k-i+1$ 种选择;当 $k+2\leqslant i<n-j+2$ 时有 $j-1$种选择;而且当 $i\geqslant n-j+2$ 时有 $n-i+1$ 种选择。

10. (a)如果 $i=2k-1$,则从$(k-1,a_i-k)$到(k,a_i-k)。若 $i=2k$,则从$(a_i-k,k-1)$到(a_i-k,k)。(b)步骤 a_{2k-1}在对角线的上部当且仅当 $k\leqslant a_{2k-1}-k$ 当且仅当$a_{2k-1}\geqslant 2k$ 当且仅当$a_{2k-1}>a_{2k}$当且仅当$a_{2k}\leqslant 2k-1$ 当且仅当 $a_{2k}-k\leqslant$

$k-1$当且仅当步骤 a_{2k}在对角线的上部。交换它们就对换了水平的和垂直的步骤。(c)步骤 a_{2k+d}在对角线之下距离至少是 m 的地方当且仅当 $k+m-1 \geqslant a_{2k+d}-(k+m)+m$ 当且仅当 $a_{2k+d}<2k+m$ 当且仅当 $a_{2k} \geqslant 2k+m$ 当且仅当 $a_{2k}-k \geqslant k+m$ 当且仅当步骤 a_{2k}在对角线之下距离至少是 m 的地方。(如果 $a_{2k+d}<2k+m$ 且 $a_{2k}<2k+m$，则至少有 $(k+m)+k$ 个元素小于 $2k+m$；这是不可能的。如果 $a_{2k+d} \geqslant 2k+m$ 且 $a_{2k} \geqslant 2k+m$，则≥中有一个必是>；但我们不能把 $\leqslant 2k+m$ 的所有元素都填入少于 $(k+m)+k$ 个位置中。因此 $a_{2k+2m-1}<a_{2k}$ 当且仅当 $a_{2k+2m-1}<2k+m$ 当且仅当 $2k+m \leqslant a_{2k}$。这是一个相当意外的结果!)

11. 考虑格子图式，即得 16 10 13 5 14 6 9 2 15 8 11 3 12 4 7 1 (61 个交换)。当 N 更大时这一情况变得更为复杂；一般地说，集合 $\{K_2, K_4, \cdots\}$ 应当是 $\{1, 2, \cdots, M-1, M, M+2, M+4, \cdots, 2\lfloor N/2 \rfloor - M\}$，排列之以便使对于 $\lfloor N/2 \rfloor$ 个元素的交换成为极大。这里 $M = \lceil 2^k/3 \rceil$ 其中 k 使 $k\lfloor N/2 \rfloor - \frac{1}{9}((3k-2)2^{k-1} + (-1)^k)$ 成为极大。交换总数的极大值是 $1 - 2\lg\lg N/\lg N + O(1/\log N)$ 乘以比较的次数。[R. Sedgewick, *SICOMP* **7** (1978), 239～272。]

12. 下列由 W. Panny 编制的程序通过指出，对于 $i = r + 2kp + s$, $k \geqslant 0$ 和 $0 \leqslant s < p$，执行步骤 M4，而避免 AND 指令。在这里，$\text{TT} \equiv 2^{t-1}$，$p \equiv \text{rI1}$，$r \equiv \text{rI2}$，$i \equiv \text{rI3}$，$i + d - N \equiv \text{rI4}$，以及 $p - 1 - s \equiv \text{rI5}$；假定 $N \geqslant 2$。

```
START   ENT1   TT              1       M1.初始化p。p←2^(t-1)
2H      ENT2   TT              T       M2.初始化q,r,d
        ST2    Q(1:2)          T       q←2^(t-1)
        ENT2   0               T       r←0
        ENT4   0,1             T       rI4←d
3H      ENT3   0,2             A       M3.对i进行循环。i←r
        INC4   -N,3            A       rI4←i+d-N
8H      ENT5   -1,1            D+E     s←0
        LDA    INPUT+1,3       C       M4.比较/交换R(i+1):R(i+d+1)
        CMPA   INPUT+N+1,4     C
        JLE    *+4             C       如果K(i+1)≤K(i+d+1)则跳转
        LDX    INPUT+N+1,4     B
        STX    INPUT+1,3       B       R(i+1)↔R(i+d+1)
        STA    INPUT+N+1,4     B
        J5Z    7F              C       如果s=p-1则跳转
        DEC5   1               C-D     s←s+1
        INC3   1               C-D     i←i+1
        INC4   1
```

```
         J4N    4B            C - D   如果 i + d < N 则重复
         JMP    5F            E       否则转到 M5
7H       INC3   1,1           D       i←i + p + 1
         INC4   1,1           D
         J4N    4B            D       如果 i + d < N 则重复循环
5H       ENT2   0,1           A       M5.对 q进行循环。 r←p
Q        ENT4   *             A       rI4←q
         ENTA   0,4           A
         SRB    1             A
         STA    Q(1:2)        A       q←q/2
         DEC4   0,1           A       rI4←d
         J4P    3B            A       如果 d≠0 则转向 M3
6H       ENTA   0,1           T       M6.对 p进行循环
         SRB    1             T
         STA    * +1(1:2)     T
         ENT1   *             T       p←⌊p/2⌋
         J1N2   2B            T       如果 p≠0 则转向 M2  ▌
```

运行时间依赖于六个量,其中仅有一个依赖于输入数据(剩下五个仅是 N 的函数):$T=t$,即"主循环"的数目;$A=t(t+1)/2$,扫描的次数或"次循环"的数目;$B=$(可变的)交换次数;$C=$比较次数;$D=$连续的比较的块区数;$E=$不完备的块区数。当 $N=2^t$ 时,不难证明 $D=(t-2)N+t+2$ 及 $E=0$。对于表 1,$T=4$,$A=10$,$B=3+0+1+4+0+0+8+0+4+5=25$,$C=63$,$D=38$,$E=0$,所以总运行时间是 $11A+6B+10C+2E+12T+1=939u$。

一般地说,当 $N=2^{e_1}+\cdots+2^{e_r}$时,Panny 已经证明,$D=e_1(N+1)-2(2^{e_1}-1)$,$E=\binom{e_1-e_r}{2}+(e_1+e_2+\cdots+e_{r-1})-(e_1-1)(r-1)$。

13.否,算法 Q 或 R 都不是。

14.(a)当 $p=1$ 时,我们对于最后的合并,进行$(2^{t-1}-0)+(2^{t-1}-1)+(2^{t-1}-2)+(2^{t-1}-4)+\cdots+(2^{t-1}-2^{t-2})=(t-1)2^{t-1}+1$ 次比较。(b)$x_t=x_{t-1}+\frac{1}{2}(t-1)+2^{-t}=\cdots=x_0+\sum_{0\leqslant k<t}\left(\frac{1}{2}k+2^{-k-1}\right)=\frac{1}{2}\binom{t}{2}+1-2^{-t}$。因此 $c(2^t)=2^{t-2}(t^2-t+4)-1$。

15.(a)考虑使 $i+d=N$ 的比较次数;然后对 r 使用归纳法。(b)如果 $b(n)=c(n+1)$,则有 $b(2n)=a(1)+\cdots+a(2n)=a(0)+a(1)+a(1)+\cdots+a(n-1)+a(n)+x(1)+x(2)+\cdots+x(2n)=2b(n)+y(2n)-a(n)$;类似地,$b(2n+1)=2b(n)+y(2n+1)$。(c)参见习题 1.2.4-42。(d)对于$(z(N)+2z(\lfloor N/2\rfloor)+\cdots)-a(N)$的一个颇为费劲的计算,并利用诸如

$\sum_{k=0}^{n}2^k(n-k)=2^{n+1}-n-2$，$\sum_{k=0}^{n}2^k\binom{n-k}{2}=2^{n+1}-\binom{n+2}{2}-1$ 的公式，即得结果

$$c(N)=N\left(\frac{1}{2}\binom{e_1}{2}+2e_1-1\right)-2^{e_1}(e_1-1)-1+\sum_{j=1}^{r}2e_j\left(e_1+\cdots+e_{j-1}-j(e_1-1)+\frac{1}{2}\binom{e_1-e_j}{2}\right)$$

16. 考虑如同图 11 和 18 中从(0,0)到(n,n)的$\binom{2n}{n}$条格子通路，而且如果 $i\geqslant j$ 便附加权$f(i-j)$，如果 $i<j$ 则附加权$f(j-i-1)+1$ 到从(i,j)到$(i+1,j)$的边上；这里 $f(k)$是在二进展开 $k=(\cdots b_2b_1b_0)_2$ 中二进位 $b_r\neq b_{r+1}$的个数。当 $N=2n$ 时在最后合并中的交换总次数为

$$\sum_{0\leqslant j\leqslant i<n}(2f(j)+1)\binom{2i-j}{i-j}\binom{2n-2i+j-1}{n-i-1}$$

R. Sedgewick 证明，对一般的 f 这个和简化成为

$\frac{n}{2}\binom{2n}{n}+2\sum_{k\geqslant1}\binom{2n}{n-k}\sum_{0\leqslant j<k}f(j)$；然后他使用伽玛函数的方法得到渐近公式

$\binom{2n}{n}(\frac{1}{4}n\lg n+(\lg\frac{\Gamma(1/4)^2}{2\pi}+\frac{1}{4}+\frac{r+2}{4\ln 2}+\delta(n))n+O(\sqrt{n}\log n)$，其中 $\delta(n)$ 是 $\lg n$ 的一个周期函数且有以.0005 为界的数量；因此当 $n\to\infty$时，平均说来，大约四分之一的比较导致交换[*SICOMP* **7** (1978), 239 ~ 272; 也参见 Flajolet 和 Odlyzko, *SIAM J. Discrete Math.* **3** (1990), 238～239]。

17. 当我们对一个 $r=N$ 且K_l 是最大键码的子文件排序时，检查 K_{N+1}。当自左至右的极小值落在位置 R_1 时，在步骤 Q9 期间，检查 K_0。

18. 在离开 Q5 之前，步骤 Q3 和 Q4 仅作对 i 和j 的一个改变；$R_l\cdots R_r$ 的分划过程在步骤 Q7 中以 $j=\lceil(l+r)/2\rceil$结束，并且尽可能完善地分开这个子文件。定量地说，我们以 $A=1,B=\lfloor(N-1)/2\rfloor,C=N+(N\bmod 2)$代替(17)；除非 $B\approx\frac{1}{2}C$，这实质上使我们处于此算法的最好情况(参看习题 27)。如果把步骤 Q3 和 Q4 的“$<$”号变为“$\leqslant$”，则本算法将不再排序；即使在(13)中我们取“$<$”号，它将交换 R_0 和 R_1，那样第三个分划阶段将把原来的 R_0 移到位置 R_2 去，等等，即成为一个真正的灾难。

19. 是，其它文件可以以任意顺序来处理。(但当每一个分划步骤相等地划分文件时队将包含 $\Omega(N/\sqrt{\log N})$个项，而可以保证栈包含的要比这小得多(参见下题)。

20. $\max(0,\lfloor\lg(N+2)/(M+2)\rfloor)$。(当 $N=2^k(M+2)-1$ 且若划分子文件时

所有的子文件皆被完全二等分,则此时出现最坏的情况。)

21. 在步骤 Q6 中恰有 t 个记录移到区域 $R_{s+1}\cdots R_N$ 中,因此 $B=t$。划分阶段以 $j=s$ 结束,因此 $C-C'=N+1-s$ 是 j 减少的次数。当诸键码不同时,在步骤 Q7 中我们也必有 $i=s+1$,因为 $i=j$ 蕴涵 $K_j=K$;于是 $C'=s$。

22. 由于 $A_{s-1}(z)A_{N-s}(z)$ 是在独立地对大小为 $s-1$ 和 $N-s$ 的随机独立有序文件排序之后,A 值的生成函数,因此容易得出 $A_N(z)$ 的所述关系。类似地,我们对于 $N>M$ 得到关系式

$$B_N(z)=\sum_{s=1}^{N}\sum_{t=0}^{s} b_{stN}z^t B_{s-1}(z)B_{N-s}(z)$$

$$C_N(z)=\frac{1}{N}\sum_{s=1}^{N} z^{N+1}C_{s-1}(z)C_{N-s}(z)$$

$$D_N(z)=\frac{1}{N}\sum_{s=1}^{N} D_{s-1}(z)D_{N-s}(z)$$

$$E_N(z)=\frac{1}{N}\sum_{s=1}^{N} E_{s-1}(z)E_{N-s}(z)$$

$$S_N(z)=\frac{1}{N}\sum_{s=1}^{N} z^{[M+1<s<N-M]}S_{s-1}(z)S_{N-s}(z)$$

这里 b_{stN} 是 s 和 t 在长度为 N 的一个文件中有给定值的概率,即

$$\binom{s-1}{t}\binom{N-s}{t}\Big/N\binom{N-1}{s-1}$$

它是 $(1/N!)$ 乘以 $(s-1)!$ 种对 $\{1,\cdots,s-1\}$ 进行排列的方法乘以 $(N-s)!$ 种对 $\{s+1,\cdots,N\}$ 进行排列的方法乘以每边上具有 t 个被移动元素的 $\binom{s-1}{t}\binom{N-s}{t}$ 种型式;对于 $0\leqslant N\leqslant M$,我们有 $B_N(z)=C_N(z)=S_N(z)=1$;$D_N(z)=\prod_{k=1}^{N}((1+(k-1)z)/k)$;而且 $E_N(z)=\prod_{k=1}^{N}((1+z+\cdots+z^{k-1})/k)$。

[当 N 很大时,考虑这些生成函数的特性是有趣的;已知类似于 $C_N(z)$ 但以 z^{N-1} 代替 z^{N+1} 的一个序列收敛到一个非正态的概率分布,但对它还未作过充分分析。参见 P. Hennequin, M. Regnier 和 U. Rosler 在 *RAIRO Theoretical Informatics and Applications* **23** (1989), 317~333; **23** (1989), 335~343; **25** (1991), 85~100 的论文。]

23. 当 $N>M$ 时,$A_N=1+(2/N)\sum_{0\leqslant k<N}A_k$;$B_N=\sum_{0\leqslant t<s\leqslant N}b_{stN}(t+B_{s-1}+B_{N-s})=(1/N)\sum_{s=1}^{N}((s-1)(N-s)/(N-1)+B_{s-1}+B_{N-s})=(N-2)/6+(2/N)\sum_{0\leqslant k<N}B_k$[参见习题 22];$D_N=(2/N)\sum_{0\leqslant k<N}D_k$;$E_N$ 是类似的;当 $N>2M+1$ 时,$S_N=(2/N)\sum_{0\leqslant k<N}S_k+(N-2M-2)/N$。对于某个函数 f_n,这些递推式的每一个都有(19)的形式。

24.对于 $N>M$，递推式 $C_N=N-1+(2/N)\sum_{0\leqslant k<N}C_k$，对于 $N>M$，有解 $(N+1)(2H_{N+1}-2H_{M+2}+1-4/(M+2)+2/(N+1))$。（所以我们节省了大约 $4N/M$ 次比较。但如果在比较之后紧跟着要有 i 同 j 的测试，则每次比较要花费更长的时间，因此除非一个键码比较的费用超过 $\frac{1}{2}M\ \ln\ N$ 乘以一个寄存器比较的费用，否则我们有损失。关于排序的许多书都没有认识到，这样一个"改进"使得快速排序显著地不快速！）。

25.（对于 $s=1$ 重复地利用(17)。）$A=N-M$，$B=0$，$C=\binom{N+2}{2}-\binom{M+2}{2}$，$D=E=S=0$。

26.实际上你不能比对 $1\ 2\ 3\cdots,N-M\ N\ N-1\cdots N-M+1$ 排序做得更糟；更微妙的答案 $N\ M-1\ M-2\cdots1\ M\ M+1\ \cdots\ N-1$ 同样是一种坏的情况。这仅仅比习题 25 坏一点，因为它使 $D=M-1$，$E=\binom{M}{2}$。

27.12　2　3　1　8　6　7　5　9　10　11　4　16　14　15　13　20　18　19　17　21　22　23，它要求 $546u$。可以证明，当通过每一个分划来分开子文件直到达到 $3M+2$ 的大小时，对于 $N=3(M+1)2^k-1$ 出现最好的情况；然后分成三份以避免压入栈的开销。我们有 $A=3\cdot2^k-1$，$C=\left(k+\frac{5}{3}\right)(N+1)$，$S=2^k-1$，$B=D=E=0$。（一般的 M 和 N 的最好情况的特性作成一个有趣但复杂的模式）。

28.递推式

$$C_n = n+1+\frac{2}{\binom{n}{3}}\sum_{k=1}^{n}(k-1)(n-k)C_{k-1}$$

可以被转换成

$$\binom{n}{3}C_n-2\binom{n-1}{3}C_{n-1}+\binom{n-2}{3}C_{n-2}=2(n-1)(n-2)+2(n-2)C_{n-2}$$

29.一般地说，考虑递推式

$$C_n = n+1+\frac{2}{\binom{n}{2t+1}}\sum_{k=1}^{n}\binom{k-1}{t}\binom{n-k}{t}C_{k-1}$$

当 $2t+1$ 个元素的均值支配分划时就发生这种情况。命 $C(z)=\sum_n C_n z^n$，这个递推式可转换成 $(1-z)^{t+1}C^{(2t+1)}(z)/(2t+2)!\ =1/(1-z)^{t+2}+C^{(t)}(z)/(t+1)!$。命 $f(x)=C^{(t)}(1-x)$；则 $p_t(\theta)f(x)=(2t+2)!\ /x^{t+2}$，其中 θ 表示算子 $x(\mathrm{d}/\mathrm{d}x)$，$p_t(x)=(t-x)^{\overline{t+1}}-(2t+2)^{\underline{t+1}}$。对于 $\alpha\neq\beta$，$(\theta-\alpha)g(x)=x^\beta$ 的通解是 $g(x)=x^\beta/(\beta-\alpha)+Cx^\alpha$；对于 $\alpha=\beta$ 是 $g(x)=x^\beta(\ln x+C)$。我们有 $p_t(-t-2)=0$；所以我们的微分方程的通解是

$$C^{(t)}(z) = (2t+2)!\ln(1-z)/p_t'(-t-2)(1-z)^{t+2} + \sum_{j=0}^{t} c_j(1-z)^{\alpha_j}$$

这里 $\alpha_0, \cdots \alpha_t$ 是 $p_t(x)=0$ 的根,而且常数 c_i 依赖于初值 $C_t, \cdots, C_{2t}$。由手边的恒等式

$$\frac{1}{(1-z)^{m+1}}\ln\left(\frac{1}{1-z}\right) = \sum_{n\geqslant 0}(H_{n+m}-H_m)\binom{n+m}{m}z^n, \quad m \geqslant 0$$

导出惊人地简单的封闭形式的解

$$C_n = \frac{H_{n+1}-H_{t+1}}{H_{2t+2}-H_{t+1}}(n+1) + \frac{1}{n!}\sum_{j=0}^{t} c_j(-\alpha_j)^{\overline{n-t}}$$

由此容易导出渐近公式。(前导项 $n \ln n/(H_{2t+2}-H_{t+1})$ 是由 M. H. van Emden [*CACM* **13** (1970), 563~567])发现的,他利用了一项信息论的方法。事实上,假设我们希望分析任何具下列性质的分划过程,它使得对于 $0 \leqslant x \leqslant 1$,当 $N \to \infty$ 时左子文件以渐近概率 $\int_0^x f(x)\,dx$ 至多包含 xN 个元素;Van Emden 已经证明,为了完整地对这个文件排序所需要的平均比较次数是近似于 $\alpha^{-1} n\ln n$,其中 $\alpha = -1/\int_0^1(f(x)+f(1-x))x\ln x\,dx$。这个公式既可应用于基数交换方法,也可用于快速排序和各种其它的方法。又见 H. Hurwitz, *CACM* **14** (1971), 99~102。)

30. *解法* 1(有历史意义的):每个子文件都可以由四个量 (l, r, k, X) 来标识,这里 l 和 r 是边界(指当前的),k 表示在整个子文件中已知具有相等的键码的字数,X 是诸键码的第 $(k+1)$ 个字的下界。采用非负的键码,我们开始时有 $(l, r, k, X) = (1, N, 0, 0)$。当分划一个文件时,我们命 K 是测试键码 K_q 的第 $(k+1)$ 个字。如果 $K>X$,则分划把所有 $\geqslant K$ 的键码划在右边,所有 $<K$ 的键码划在左边。(每次仅仅考察键码的第 $(k+1)$ 个字);通过分划产生的子文件的标识分别是 $(l, j-1, k, X)$ 和 (j, r, k, K)。但如果 $K=X$ 则分划把所有 $>K$ 的键码划在右边,所有 $\leqslant K$[实际上 $=K$]的键码划在左边,通过分划产生的子文件的标识分别是 $(l, j, k+1, 0)$ 以及 $(j+1, r, k, K)$。在两种情况下,都不能保证 R_j 是在它最后的位置,因为我们没有考察第 $(k+2)$ 个字。为了适当处理边界条件,应作显然的进一步的修改。通过增加第五个"上限"分量,这个方法可以成为对左边和右边是对称的。

解法 2,由 Bentley 和 Sedgewick 给出[*SODA* **8** (1997), 360~369]:在由 (l, r, k) 标识的一个子文件中,和在解法 1 中一样命 K 是 K_q 的字 $k+1$,但使用习题 41 的算法来把子文件分成对于 $<K, =K, >K$ 的三种情况的 $(l, i-1, k), (i, j, k+1), (j+1, r, k)$。作者们称这个方法为*多键码快速排序*,它比解法 1 显著地好得多,因此它堪同对字符串进行排序的最快的方法相匹敌。

31. 通过一个正常的分划过程使 R_1 最后落到 R_s 上。如果 $s=m$,则停止;如果 $s<m$,则使用同样的技术来求右边子文件的第 $(m-s)$ 个最小的元素;如果 $s>m$ 则找左边子文件的第 m 个最小元素[*CACM* **4** (1961), 321~322; **14** (1971), 39~45]。

R. G. Dromey[*Sofeware Practice & Experience*, **16** (1986), 981～986]已经注意到，只要 i 或 j 已经达到位置 m 时，如果我们停止每个分划，就需要较少的比较和交换。

32. 递推式是 $C_{11}=0$ 和当 $n>1$ 时，$C_{nm}=n+1+(A_{nm}+B_{nm})/n$，其中对于 $1\leqslant m\leqslant n$

$$A_{nm}=\sum_{1\leqslant s<m}C_{(n-s)(m-s)} \quad 和 \quad B_{nm}=\sum_{m<s\leqslant n}C_{(s-1)m}$$

由于 $A_{(n+1)(m+1)}=A_{nm}+C_{nm}$ 和 $B_{(n+1)m}=B_{nm}+C_{nm}$，因此我们可以首先求量 $D_n=(n+1)C_{(n+1)(m+1)}-nC_{nm}$ 的一个公式，然后对它们求和以得到答案 $2((n+1)H_n-(n+2-m)H_{n+1-m}-(m+1)H_m+n+\frac{5}{3})-\frac{1}{3}\delta_{mn}-\frac{1}{3}\delta_{m1}-\frac{2}{3}\delta_{mn}\delta_{m1}$。当 $n=2m-1$ 时，它变成为 $4m(H_{2m-1}-H_m)+4m-4H_m+\frac{4}{3}(1-\delta_{m1})=(4+4\ln 2)m-4\ln m-4\gamma-\frac{5}{3}+O(m^{-1})\approx 3.39n$。[参见 D. E. Knuth, *Proc. IFIP Congress* (1971), 19～27]。

另一个解法由 6.2.2 节的理论得出：假设诸键码是{1,2,…,n}，且设 X_{jk} 是在对应于快速排序的二分查找树中节点 j 和 k 公共的祖先的个数。于是可以证明，由习题 31 的算法所作的比较次数为 $\sum_{j=1}^{n}X_{jm}+X_{mm}-2$[节点 m 是一个叶]。在一个随机二分查找树中节点 i 是节点 j 和 k 公共的祖先的概率是 $1/(\max(i,j,k)-\min(i,j,k)+1)$。由下列事实，即对于 $1\leqslant j\leqslant k$，$EX_{jk}=H_j+H_{n+1-k}+1-2H_{k-j+1}$，以及 Pr(节点 m 是一个叶) = Pr(在一个随机排列中 m 之后不是 $m\pm 1$) $=\frac{1}{3}+\frac{1}{6}\delta_{m1}+\frac{1}{6}\delta_{mn}+\frac{1}{3}\delta_{m1}\delta_{mn}$。[参见 R. Raman, *SIGACT News* **25**, 2(1994 年 6 月), 86～89]。

关于使用三取中的分划的一个类似选择算法的分析，参见 Kirschenhofer, Prodinger 以及 Martínez, *Random Structures and Algorithms* **10**(1997), 143～156。在习题 5.3.3-24 中讨论了一些渐近地更快的方法。

33. 像在基数交换的第一阶段那样进行，并且利用正负号代替二进位 1。

34. 只要我们在每个阶段已经找到至少一个二进位 0 和至少一个二进位 1，即在每个阶段中进行了头一次交换之后，即可以避免作是否 $i\leqslant j$ 的判断，这在程序 R 中节省了接近 $2C$ 的时间单位。

35. $A=N-1$, $B=(\min 0, \text{ave } \frac{1}{4}N\lg N, \max \frac{1}{2}N\lg N)$, $C=N\lg N$, $G=\frac{1}{2}N$, $K=L=R=0$, $S=\frac{1}{2}N-1$, $X=\left(\min 0, \text{ave } \frac{1}{2}(N-1), \max N-1\right)$。一般来说，量 A,C,G,K,L,R 和 S 仅依赖于文件中的键码的集合，而不依赖于它们开始时的次序，仅仅 B 和 X 受键码的初始顺序的影响。

36.(a)$\sum\binom{n}{k}\binom{k}{j}(-1)^{k+j}a_j=\sum\binom{n}{j}\binom{n-j}{k-j}(-1)^{k-j}a_j=\sum\binom{n}{j}\delta_{nj}a_j=a_n$。

(b)$\langle\delta_{n0}\rangle$;$\langle-\delta_{n1}\rangle$;$\langle(-1)^m\delta_{nm}\rangle$;$\langle(1-a)^n\rangle$;$\left\langle\binom{n}{m}(-a)^m(1-a)^{n-m}\right\rangle$。(c)把有待证明的关系式写成 $x_n=y_n=a_n+z_n$,由(a)部分我们有 $y_n=a_n+z_n$;而且 $2^{1-n}\sum_{k\geqslant 2}\binom{n}{k}y_k=z_n$,所以 y_n 和 x_n 一样满足相同的递推式。[关于这个结果的某些推广,见习题 53 和 6.3-17。直接证明 $\hat{x}_n=\hat{a}_n2^{n-1}/(2^{n-1}-1)$看来并不容易。]

37. $\left\langle\sum_m c_m\binom{n}{2m}2^{-n}\right\rangle$,其中 $c_0,c_1,c_2,\cdots$为任意常数序列。[这个答案尽管是正确的,但并不立即揭示出$\langle 1/(n+1)\rangle$和$\langle n-\delta_{n1}\rangle$是这样的序列!形式为$\langle a_n+\hat{a}_n\rangle$的序列总是自对偶的。注意,借助于生成函数 $A(z)=\sum a_nz^n/n!$ 我们有 $\hat{A}(z)=e^zA(-z)$;因此 $A=\hat{A}$ 等价于说,$A(z)e^{-z/2}$是一个偶函数。]

38.产生大小为 s 的一个左文件和大小为$N-s$ 的一个右文件的一个分划阶段,对总的运行时间作出如下的贡献:

$$A=1,\quad B=t,\quad C=N,\quad K=\delta_{s1},\quad L=\delta_{s0},\quad R=\delta_{sN},\quad X=h,$$

这里 t 是$K_1,\cdots,K_s$ 诸键码中二进位b 等于 1 的键码的个数,而且 h 是K_{s+1}的二进位 b;如果 $s=N$,则 $h=0$(参见(17))。这导致诸如

$$B_N=2^{-N}\sum_{0\leqslant t\leqslant s\leqslant N}\binom{s}{t}\binom{N-s}{t}(t+B_s+B_{N-s})=$$

$$\frac{1}{4}(N-1)+2^{1-N}\sum_{s\geqslant 2}\binom{N}{s}B_s,\quad \text{对于 } N\geqslant 2;B_0=B_1=0$$

的递推方程(参见习题 23)。用习题 36 的方法来解这些递推式,得出公式 $A_N=V_N-U_N+1$,$B_N=\frac{1}{4}(U_N+N-1)$,$C_N=V_N+N$,$K_N=N/2$,$L_N=R_N=\frac{1}{2}(V_N-U_N-N)+1$,$X_N=\frac{1}{2}A_N$。显然 $G_N=0$。

39.快速排序的每个阶段至少把一个元素放入到它最后的位置,但在基数交换期间则不一定(参见表 3)。

40.如果我们在步骤 R2 中每当 $r-l<M$ 时即转到直接插入,则这个问题不出现,除非有多于 M 个相等的元素出现。如果后者是一种可能的前景,则在步骤 R8 中每当 $j<l$ 或$j=r$ 时,我们可以判断是否 $K_l=\cdots=K_r$。

41.Lutz M. Wegner[*IEEE Trans* **C-34**(1985),362~367]讨论了若干种方法,其中下面这个(由 Bentley 和 McIlroy 在*Software Practice & Exp.* **23**(1993),1256~1258 作了简化)方法看来在实践中是最好的。基本的思想是以五部分的数组

= K	< K	?	> K	= K

$l \quad a \quad b \quad c \quad d \quad r$

来进行工作直到中间部分为空为止，然后把两端交换到中间去。

D1.［初始化］置 $a \leftarrow b \leftarrow l, c \leftarrow d \leftarrow r$。

D2.［增加 b 直到 $K_b > K$ 为止］如果 $b \leqslant c$ 和 $K_b < K$，则 b 加 1 并且重复这一步骤。如果 $b \leqslant c$ 且 $K_b = K$，则交换 $R_a \leftrightarrow R_b$，a 和 b 加 1，并且重复这一步骤。

D3.［减少 c 直到 $K_c < K$ 为止］如果 $b \leqslant c$ 和 $K_c > K$，则 c 减 1 并且重复这一步骤。如果 $b \leqslant c$ 且 $K_c = K$，则交换 $R_c \leftrightarrow R_d$，c 和 d 减 1，并重复这一步骤。

D4.［交换］如果 $b < c$，交换 $R_b \leftrightarrow R_c$，b 加 1，c 减 1，并返回 D2。

D5.［清除］对于 $0 \leqslant \mathrm{k} < \min(a-l, b-a)$ 交换 $R_{l+k} \leftrightarrow R_{c-k}$。对于 $0 \leqslant k < \min(d-c, r-d)$ 也交换 $R_{b+k} \leftrightarrow R_{r-k}$。最后置 $i \leftarrow l+b-a, j \leftarrow r-d+c$。▌

对于步骤 D1 的直截了当的修改将有效地处理蜕化情况并且确保在我们达到 D2 之前，有 $a < b$ 和 $c < d$。然后在 D2 和 D3 中对"$b \leqslant c$"的测试就不必要了；参见习题 24。而且，这个变动将使这些步骤本身无需交换记录。

排序的主要应用之一是把有相等键码的记录放在一起，因此算法 Q 的三划分方案通常比二划分方案更可取。在步骤 D5 中的交换是有效的因为具有等于 K 的键码的所有记录现在都在它们最后的栖息地。

本习题是由 W.H.J,Feijen 给出的，他把它叫做"荷兰国旗问题"：给定在一列中随机地安排的红，白和蓝的标记的集合，判定应如何交换标记对使得红的标记将居于顶上，而蓝的那些全都居于下边，同时将只考察每个标记一次和仅使用一些辅助变量来控制这个过程［参见 E. W. Dijkstra, *A Discipline of Programming* (Prentice-Hall, 1976)，第 14 章］。

42. 这是由 R.M.Karp 给出的一般定理的特殊情况；参见*JACM* **41**(1994)，1136～1150，§2.8。McDiarmid 和 Hayward, *J. Algorithms* **21**(1996)，476～507 已经得到对于快速排序分布的尾部更准确得多的渐近上限。

43. 当 $a \to 0+$ 时由习题 1.2.7-24，$\int_0^1 y^{a-1}(e^{-y}-1)dy + \int_1^\infty y^{a-1}e^{-y}dy = \Gamma(a) - 1/a = (\Gamma(a+1) - \Gamma(1))/a \to \Gamma'(1) = -\gamma$。

44. 对于 $k \geqslant 0$，我们有 $r_k(m) \sim \frac{1}{2}(2m)^{(k+1)/2}\Gamma((k+1)/2) - \delta_{ko} - \sum_{j \geqslant 0}(-1)^j B_{k+2j+1}/((k+2j+1)j!\,(2m)^j)$。当 $k=-1$ 时(36)中来自 $f_k^{(j-1)}(m)$ 的贡献与 H_{m-1} 的展开式中类似的项相消，而且我们有 $r_{-1}(m) = H_{m-1} + (1/\sqrt{2}\,m)\sum_{t \geqslant 0} f_{-1}(t) \sim \frac{1}{2}(\ln(2m) + \gamma) - \sum_{j \geqslant 1}(-1)^j B_{2j}/(2j)j!\,(2m)^j$。因此由(33)的项

N^t/t 对 W_{m-1} 的贡献得自和数 $m\sum_{t\geqslant1}t^{-1}\exp(-t^2/2m)(1-t^3/3m^2+t^6/18m^4)(1-t^4/4m^3)(1-t/2m-t^2/8m^2)+O(m^{-1/2})=\frac{1}{2}m\ln m+\frac{1}{2}(\ln2+\gamma)m-\frac{5}{12}\sqrt{2\pi m}+\frac{4}{9}+O(m^{-1/2})$。项 $-\frac{1}{2}N^{t-1}$ 贡献 $-\frac{1}{2}\sum_{t\geqslant1}\exp(-t^2/2m)(1-t^3/3m^2)(1-t/2m)$、$(1-t/m)+O(m^{-1/2})=-\frac{1}{4}\sqrt{2\pi m}+\frac{1}{3}$。项 $\frac{1}{2}\delta_{t1}$ 产生 $\frac{1}{2}$。最后项 $\frac{1}{2}(t-1)B_2N^{t-2}$ 贡献 $\frac{1}{12}m^{-1}\sum_{t\geqslant1}t\exp(-t^2/2m)+O(m^{-1/2})=\frac{1}{12}+O(m^{-1/2})$。

45. 用来推导(42)的论证对于(43)也是有效的,但应略去在 $z=-1$ 和 $z=0$ 处的残数。

46. 如同我们对于(45)所做的那样,我们得到 $(s-1)!\,/\ln 2+\delta_s(n)$,这里

$$\delta_s(n)=\frac{2}{\ln 2}\sum_{k\geqslant1}\mathscr{R}(\Gamma(s-2\pi ik/\ln 2)\exp(2\pi ik\lg n))$$

(注意:对于整数 $s\geqslant0$,$|\Gamma(s+it)|^2=(\prod_{0\leqslant k<s}(k^2+t^2))\pi/(t\sinh\pi t)$,所以我们可以给出 $\delta_n(s)$ 的界。)

47. 事实上,对于所有 $s>0$,$\sum_{j\geqslant1}e^{-n/2^j}(n/2^j)^s$ 等于习题 46 中的积分。

48. 利用中间恒等式

$$1-e^{-x}=\frac{-1}{2\pi i}\int_{-1/2-i\infty}^{-1/2+i\infty}\Gamma(z)x^{-z}dz$$

我们沿用正文中的方法,但 $1-e^{-x}$ 起着 $e^{-x}-1+x$ 的作用;$V_{n+1}/(n+1)=(-1/2\pi i)\int_{-\frac{1}{2}-i\infty}^{-\frac{1}{2}+i\infty}\Gamma(z)n^{-z}dz/(2^{-z}-1)+O(n^{-1})$,而且在习题 46 的记号下,这个积分等于 $\lg n+\gamma/\ln 2-\frac{1}{2}-\delta_0(n)+O(n^{-100})$。[于是在习题 38 中的量 A_N 是 $N(1/\ln 2-\delta_0(N-1)-\delta_{-1}(N))+O(1)$。]

49. 等式(40)的右边可以改进为估计 $e^{-x}\left(1-\frac{1}{2}x^2/h+O((x^3+x^4)n^{-2})\right)$。其效果是使习题 47 中的和减半,并且以 $2-\frac{1}{2}(1/\ln 2+\delta_1(n))+O(n^{-1})$ 代替(50)中的 $O(1)$。("2"来自于(46)中的"$2/n$"。)

50. $U_{mn}=n\log_m n+n((\gamma-1)/(\ln m)-\frac{1}{2}+\delta_{-1}(n))+m/(m-1)-1/(2\ln m)-\frac{1}{2}\delta_1(n)+O(n^{-1})$,其中 $\delta_s(n)$ 像在习题 46 中那样定义,但以 $\ln m$ 和 $\log m$ 来代替 $\ln 2$ 和 lg[注意:对于 $m=2,3,4,5,10,100,1000,10^6$,我们分别有 $\delta_{-1}(n)<.000000172501,.000041227,.0002963,.0008501433,.0062704,.06797,.1525,.348$]。

51. 设 $N=2m$。我们可以推广和式(35)到所有 $t\geqslant 1$,那时它等于

$$\sum_{t\geqslant 1}\frac{1}{2\pi i}\int_{a-i\infty}^{a+i\infty}\Gamma(z)(t^2/N)^{-z}t^k dz=$$
$$\frac{1}{2\pi i}\int_{a-i\infty}^{a+i\infty}\Gamma(z)N^z\zeta(2z-k)dz$$

假定 $a>(k+1)/2$。所以我们需要知道 zeta 函数的性质。当 $\mathscr{R}(w)\geqslant -q$ 且 $|w|\to\infty$ 时 $\zeta(w)=O(|w|^{q+1})$;因此如果仅考虑残数,则我们可以随心所欲地向左移动积分的边。因子 $\Gamma(z)$ 在 $0,-1,-2,\cdots$ 处有极点,$\zeta(2z-k)$ 仅在 $z=(k+1)/2$ 处有一个极点。在 $z=-j$ 处的残数是 $N^{-j}(-1)^j\zeta(-2j-k)/j!$ 以及 $\zeta(-n)=(-1)^nB_{n+1}/(n+1)$。在 $z=(k+1)/2$ 处的残数是 $\frac{1}{2}\Gamma((k+1)/2)N^{(k+1)/2}$。但当 $k=-1$ 时,在 $z=0$ 处有一个双重极点;而且 $\zeta(z)=1/(z-1)+\gamma+O(|z-1|)$,所以在这种情况下 0 点处的残数为 $\gamma+\frac{1}{2}\ln N-\frac{1}{2}\gamma$。我们因此得到在习题 44 的答案中所述的渐近级数。

52. 置 $x=t/n$;则

$$\binom{2n}{n+t}\bigg/\binom{2n}{n}=\exp(-2n(x^2/1\cdot 2+x^4/3\cdot 4+\cdots)+(x^2/2+x^4/4+\cdots)-$$
$$(1/6n)(x^2-x^4+\cdots)+\cdots)$$

对于各种的 k 现在可以借助于 $\sum t\geqslant 1t^k d(t)e^{-t^2/n}$ 来表达所求的和数。沿用习题 51 的方法,由于 $\zeta(z)^2=\sum t\geqslant 1d(t)t^{-z}$,我们希望来计算当 $k\geqslant 0$ 时 $\Gamma(z)n^z\zeta(2z-k)^2$ 的残数。在 $z=-j$ 处,这个残数是 $n^{-j}(-1)^j(B_{2j+k+1}/(2j+k+1))^2/j!$,而在 $z=(k+1)/2$ 处,这个残数是 $n^{(k+1)/2}\Gamma((k+1)/2)\left(\gamma+\frac{1}{4}\ln n+\frac{1}{4}\psi((k+1)/2)\right)$,其中 $\psi(z)=\Gamma'(z)/\Gamma(z)=H_{z-1}-\gamma$;于是,例如,当 $k=0$ 时对所有 M 有 $\sum_{t\geqslant 1}e^{-t^2/n}d(t)=\frac{1}{4}\sqrt{\pi n}\ln n+\left(\frac{3}{4}\gamma-\frac{1}{2}\ln 2\right)\sqrt{\pi n}+\frac{1}{4}+O(n^{-M})$。对于 $S_n\bigg/\binom{2n}{n}$,把 $\left(\frac{1}{32}\ln n+\frac{3}{32}\gamma+\frac{1}{24}-\frac{1}{16}\ln 2\right)\sqrt{\pi/n}+O(n^{-1})$ 加到这个量上(参见习题 1.2.7-23 和 1.2.9-19)。

53. 设 $q=1-p$,推广习题 36(c),如果

$$x_n=a_n+\sum_{k\geqslant 2}\binom{n}{k}(p^kq^{n-k}+q^kp^{n-k})x_k$$

则

$$x_n=a_n+\sum_{k\geqslant 2}\binom{n}{k}(-1)^k\hat{a}_k(p^k+q^k)/(1-p^k-q^k)$$

因此我们可以像以前那样求 B_N 和 C_N;B_N 中的因子 $\frac{1}{4}$ 应以 pq 代替。U_N 的渐近性

质可基本上如正文中那样考察,而且

$$T_n = \sum_{r\geqslant 1,s\geqslant 0}\binom{r}{s}(e^{-np^sq^{r-s}} - 1 + np^sq^{r-s}) =$$

$$\frac{1}{2\pi i}\int_{-3/2-i\infty}^{-3/2+i\infty}\Gamma(z)n^{-z}(p^{-z}+q^{-z})dz/(1-p^{-z}-q^{-z}) =$$

$$(n/h_p)(\ln n + \gamma - 1 + h_p^{(2)}/2h_p - h_p + \delta(n)) + O(1)$$

其中

$$h_p = -(p\ln p + q\ln q), h_p^{(2)} = p(\ln p)^2 + q(\ln q)^2$$

而且

$$\delta(n) = \sum \Gamma(z)n^{-1-z}/h_p$$

这里求和对所有使得 $p^{-z}+q^{-z}=1$ 的复数 $z\neq 1$ 进行。后一点集一般地似乎是难于分析的;但当 $p=\phi^{-1}, q=\phi^{-2}$时,解为 $z=(-1)^{k+1}+k\pi i/\ln\phi$。主项$(n\ln n)/h_p$也可以从习题 29 的解答中所引的 van Emden 的一般公式得到。对于 $p=\phi^{-1}$,我们有 $1/h_p=1.503718$,相对于 $1/h_{1/2}\approx 1.442695$。

54. 命 C 是半径为$\left(M+\frac{1}{2}\right)b$ 的一个圆,于是当 $M\to\infty$时在 C 上的积分变为0。(U_N 的渐近形式现在可以用一种新方法即展开 $\Gamma(n+1)/\Gamma(n+ibm)$来导出。当 f 有相当好的特性时,本题的方法可应用于形如

$$\sum_k\binom{n}{k}(-1)^{n-k}f(k) = \frac{-1}{2\pi i}\oint B(n+1,-z)f(z)dz$$

的所有和。后一公式可在 N. E. Nörlund 的*Vorlesungen über Differen-zenrechnung*(柏林:Springer,1924),103 中找到。)

55. 以下列程序段代替程序 Q 的行 04~06,后边再接上‘STA INPUT+1,2’(参见(27)之后的注释):

```
2H  ENTA  0,2
    INCA  0,3
    SRB   1
    STA   *+1(0:2)
    ENT4  *
    LDA   INPUT,2     rA←a
    LDX   INPUT,3     rX←c
    CMPA  INPUT,3
    JL    1F
    CMPA  INPUT,4     rA:b
    JLE   3F
    CMPX  INPUT,4     rX:b
    JG    4F
    STA   INPUT,3     c≤b<a
    STX   INPUT,2
5H  LDA   INPUT,4     rA←b
    JMP   6F
4H  LDA   INPUT,3     b<c≤a
    LDX   INPUT,2
    STX   INPUT,3
    JMP   5F
3H  STX   INPUT,2     c≤a≤b
    LDX   INPUT,4
    STX   INPUT,3
    JMP   6F
1H  CMPA  INPUT,4
    JGE   5F
    CMPX  INPUT,4     a<b,c
    JGE   5B
    LDA   INPUT,3     a<c<b
    LDX   INPUT,4
    STX   INPUT,3
    JMP   6F
5H  LDX   INPUT,4     b≤a<c
    STX   INPUT,2
6H  LDX   INPUT+1,2
    STX   INPUT,4
    ENT4  2,2
    ENT5  0,3
```

并把行 22 的指令改成为“STX INPUT+1,2”。如果二进移位不能用,则头三条指令须改成“ENTX 0,2;INCX 0,3;ENTA 0;DIV=2=”。

这一程序实质上交换 R_{l+1} 和 $R_{\lfloor (l+r)/2\rfloor}$ 并对三个记录 R_l, R_{l+1}, R_r 进行排序，然后应用通常的分划到 $R_{l+1}\cdots R_{r-1}$ 上。通过简单地把中间元素放在 rA 中，把 R_l 传送到中间元素以前的位置，并且如同它现在这样来使用程序 Q 来节省代码的一些行，是吸引人的。但这样一个方法有坏结果，因为它需要 N^2 步的阶来对文件 N $N-1\cdots 1$ 进行排序。(这一惊奇的结果，首先是由 D. B. Coldrick 发现的，要见到你才能信——试试看!)上边推荐的这一技术，是属于 R. Sedgewick 的，看来避免了这种"简单的最坏情况"的异常，而且运行也更快。

通过这个三个取中的分划方案，这个算法不考察 K_{N+1}，但在步骤 Q9 中它们可能考察 K_0。

56. 通过命 $y_n = nx_n$, $u_n = ny_{n+1} - (n+2)y_n$, $v_n = nu_{n+1} - (n-5)u_n$；对于 $n > m$，我们可以解递推式 $\binom{n}{3}x_n = b_n + 2\sum_{k=1}^{n}(k-1)(n-k)x_{k-1}$。由此得出对于 $n > m$, $v_n = 6(b_{n+2} - 2b_{n+1} + b_n)$。例如：对于 $n \leqslant m$，命 $x_n = \delta_{n1}$，且设 $b_n \equiv 0$。则对于所有的 $n > m$, $v_n = 0$，因此 $n^{\underline{5}} - u_{n+1} = m^{\underline{5}} - u_{m+1}$。由于 $y_{m+1} = 12/m$, $y_{m+2} = 12/(m+1)$，我们最终求得对于 $n > m$, $x_n = \frac{48}{7}(n+1)/m(m+1)(m+2) + \frac{36}{7}(m-1)^{\underline{4}}/n^{\underline{6}}$。一般地，命 $f_n = (12/(n-1)(n-2))\sum_{k=1}^{n}(k-1)(n-k)x_{k-1}$；当 b_n 恒等于 0 时对于 $n > m$ 的解是

$$x_n = (n+1)\frac{(m+1)f_{m+2} - (m-4)f_{m+1}}{7(m+1)(m+2)} - \frac{((m+1)f_{m+2} - (m+3)f_{m+1})m^{\underline{5}}}{7n^{\underline{6}}}$$

对于 $n \leqslant m$，当 $b_n = \binom{n}{3}/n^{\underline{p}}$ 和 $x_n = 0$ 时，这个解是

$$\frac{x_n}{n+1} = \frac{(p-3)(p-2)}{(p-6)(p+1)(n+1)^{\underline{p+1}}} + \frac{12}{7}\frac{1}{(p+1)(m+2)^{\underline{p+1}}} - \frac{12}{7}\frac{(m+1-p)^{\underline{6-p}}}{(p-6)(n+1)^{\underline{7}}}$$

对于 $n > m$；除非当 $p = -1$ 时，我们有 $x_n/(n+1) = \frac{12}{7}(H_{n+1} - H_{m+2}) + \frac{37}{49} + \frac{12}{49}(m+2)^{\underline{7}}/(n+1)^{\underline{7}}$，而且当 $p = 6$ 时，$x_n/(n+1) = -\frac{12}{7}(H_{n-6} - H_{m-5})/(n+1)^{\underline{7}} + \frac{12}{49}/(m+2)^{\underline{7}} + \frac{37}{49}/(n+1)^{\underline{7}}$。

像在习题 21-23 中那样论证，我们发现头一个分划阶段现在对 A 贡献 1，对 B 贡献 t，对 C 贡献 $N-1$，其中 t 如同以前定义，但在做了习题 55 中的重新安排之后。在这个新的假定之下，我们求得 $b_{stN} = 6\binom{s-2}{t}\binom{N-s-1}{t}\Big/N\binom{N-1}{s-1}$；因此上面所述的递推式以下列方式出现：

	对于 $N\leqslant M$ 的值	对于 $N>M$ 的 $b_N/\binom{N}{3}$	对于 $N>M$ 的解
A_N	0	1	$(N+1)\left(\frac{12}{7}/(M+2)\right)-1+O(N^{-6})$
B_N	0	$(N-4)/5$	$(C_N-3A_N)/5$
C_N	0	$N-1$	$(N+1)\left(\frac{12}{7}(H_{N+1}-H_{M+2})+\frac{37}{49}-\frac{24}{7}/(M+2)\right)+2+O(N^{-6})$
D_N	$N-H_N$	0	$(N+1)\left(1-\frac{12}{7}H_{M+1}/(M+2)-\frac{4}{7}/(M+2)\right)+O(N^{-6})$
E_N	$N(N-1)/4$	0	$(N+1)\left(\frac{6}{35}M-\frac{17}{35}+\frac{6}{7}/(M+2)\right)+O(N^{-6})$

类似地，$S_N=\frac{3}{7}(N+1)(5M+3)/(2M+3)(2M+1)-1+O(N^{-6})$。在习题 55 中的程序总共的平均运行时间是 $53\frac{1}{2}A_N+11B_N+4C_N+3D_N+8E_N+9S_N+7N$；$M=9$ 的选择比 $M=10$ 的选择还要好些，并产生近似于 $10\frac{22}{35}N\ln N+2.116N$ 的平均时间[*Acta Inf.* **7**(1977)，336～341]。若以 DIV 代替 SRB，则运行时间增加 $11A_N$ 并取 $M=10$。

5.2.3 小节

1. 否；考虑 $K_1>K_2=\cdots=K_N$ 的情况。但使用∞的方法(在算法 S 的紧前边描述之)是稳定的。

2. 如果我们从较高的下标到较低的下标，来扫描在内存中顺序地存放的一个线性表，则这种遍历方法通常都稍微快些，因为判断一个下标是否为 0 通常比起判断它是否超过 N 更容易些。(由于同一原因，步骤 S2 的查找从 j 往下运行到 1；但见习题 8！)

3. (a)对于 $N\,a_2\cdots a_{N-1}a_1$，$a_1\,N\,a_3\cdots a_{N-1}a_2$，$\cdots$，$a_1a_2\cdots$，$a_{N-2}N\,a_{N-1}$，$a_1\cdots a_{N-1}N$ 等输入，出现排列 $a_1\cdots a_{N-1}N$。(b)如同在 1.2.10 节中所示那样，在步骤 S2 的头一次迭代期间，极大值改变的平均次数是 H_N-1。[因此，B_N 可以从等式 1.2.7-(8)求得]。

4. 如果输入是$\{1,2,\cdots,N\}$的一个排列，则在步骤 S3 中 $i=j$ 的次数恰比排列中的轮换个数少 1。(其实，不难证明，步骤 S2 和 S3 只不过从它的轮换中撤销元素 j；因此 S3 仅当 j 是它的轮换中最小元素时才什么也不干)。由等式 1.3.2-(21)，平

均说来,我们可以节省步骤 S3 的 $N-1$ 次执行中的 H_N-1 次。

于是,在步骤 S3 之前插入一个额外的判断"$i=j$?"是低效的。然而我们可以不去判断 i 和 j 的值,而是稍微地加长 S2 的程序,重复一部分代码,使得如果在查找极大值期间初始的猜测 K_j 不改变,则决不会遇到 S3;这将使程序 S 稍微快一点。

5. $(N-1)+(N-3)+\cdots=\lfloor N^2/4\rfloor$。

6. (a)在步骤 S3 中如果 $i\neq j$,则该步使反序数减少 $2m-1$,其中 m 比 $K_{i+1}\cdots K_{j-1}$诸键码中那些处于 K_i 和 K_j 之间键码的个数大 1;显然 m 不少于上一个步骤 S2 中对 B 的贡献。现在应用习题 4 的观察,把循环同条件 $i=j$ 联系起来。(b)通过逐次的交换反序的相邻元素,每个排列均可从 $N\cdots21$ 得到。(以相反的顺序应用把排列排成递减次序的交换。)每个这样的操作使 I 减 1,并且使 C 变化 ±1。因此,没有任何排列有 $I-C$ 的一个值,这个值超过 $N\cdots2\ 1$ 的相应值。[由习题 5,不等式 $B\leqslant\lfloor N^2/4\rfloor$是最好的。]

7. 姚期智,"On Straight Selection Sort",计算机科学技术报告 185(普林斯顿大学,1988)证明,方差是 $\alpha N^{1.5}+O(N^{1.495}\log N)$,其中 $\alpha=\frac{4}{3}\sqrt{\pi}\ln\frac{4}{e}\approx0.9129$;他也猜测,实际的误差项要小得多。

8. 假定我们已经记住了 $\max(K_1,\cdots,K_{i-1})$,则可以在位置 K_i 处开始步骤 S2 的下一次迭代。记住所有这些辅助信息的一个方法是使用一个链接表 $L_1\cdots L_N$,使得只要 K_k 是黑体的,则 K_{l_k} 是上一个黑体元素;$L_1=0$。[我们以某些冗余的比较为代价,也可以减少所用的辅助存储。]

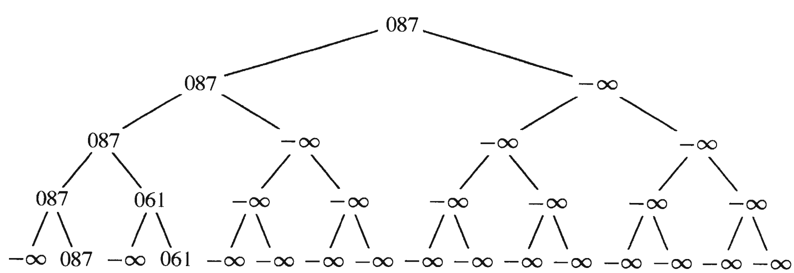
下列的 MIX 程序使用地址修改,使得内循环加快。$rI1\equiv j$, $rI2=k-j$, $rI3\equiv i$, $rA\equiv K_i$。

```
START   ENT1   N             1          j←N
        STZ    LINK+1        1
        JMP    9F            1
1H      ST1    6F(0:2)       N−D        修改循环中的地址
        ENT4   INPUT,1       N−D
        ST4    7F(0:2)       N−D
        ENT4   LINK,1        N−D
        ST4    8F(0:2)       N−D
7H      CMPA   INPUT+J,2     A                              [修改地址]
        JGE    *+4           A          如果 Ki≥Kk 则转移
8H      ST3    LINK+J,2      N+1−C      否则 Lk←i           [修改地址]
6H      ENT3   J,2           N+1−C      i←k                 [修改地址]
        LDA    INPUT,3       N+1−C
        INC2   1             A          k←k+1
        J2NP   7B            A          如果 k≤j 则转移
```

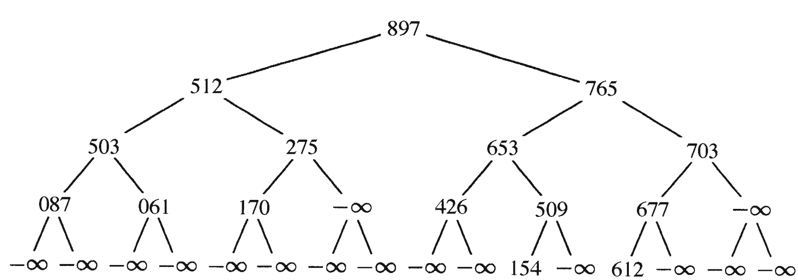
```
4H   LDX    INPUT,1    N
     STX    INPUT,3    N      Ri←Rj
     STA    INPUT,1    N      Rj←以前的 Ri
     DCE1   1          N      j←j-1
     ENT2   0,3        N      rI2←i
     LD3    LINK,3     N      i←Li
     J3NZ   5F         N      如果 i>0 则 k 将在 i 处开始
9H   ENT3   1          C      否则 i←1
     ENT2   2          C      k 将在 2 处开始
5H   DEC2   0,1        N+1
     LDA    INPUT,3    N+1    rA←Ki
     J2NP   1B         N+1    如果 k≤j 则转移
     J1P    4B         D+1    如果 j>0 则转移   ▌
```

9. $N-1+\sum_{N\geqslant k\geqslant 2}((k-1)/2-1/k)=\frac{1}{2}\binom{N}{2}+N-H_N$。[$C$ 和 D 的平均值分别为 H_N+1 和 $H_N-\frac{1}{2}$；因此这个程序的平均运行时间为 $(1.25N^2+31.75N-15H_N+14.5)u$。]程序 H 要好得多。

10.



11.



12. 对于每个分支节点中的 $-\infty$ 1 次，共 2^n-1 次。

13. 如果 $K \geqslant K_{r+1}$，则当 $j=r$ 时步骤 H4 可以转到步骤 H5。(除非 $K_r < K_{r+1}$，步骤 H5 什么也不干，这时步骤 H6 无论如何将转到 H8)。为确保在整个算法中 $K \geqslant K_{r+1}$，我们可以以 $K_{N+1} \leqslant \min(K_1, \cdots, K_N)$ 开始；在步骤 H2 中不置 $R_r \leftarrow R_1$，而置 $R_{r+1} \leftarrow R_{N+1}$ 和 $R_{N+1} \leftarrow R_1$；在 $r=1$ 之后还置 $R_2 \leftarrow R_{N+1}$。(这项技巧既不加快这个算法也不使程序 H 有任何缩短)。

14. 当插入一个元素时，给它一个键码，这个键码小于(或大于)所有以前指定的键码，以达到一个简单队(或栈，分别地)的效果。

15. 为了有效起见，下列的解是有一点技巧的，它避免了 3 的所有倍数 [*CACM* **10**(1967),570]。

P1. 置 $p[1] \leftarrow 2, p[2] \leftarrow 3, k \leftarrow 2, n \leftarrow 5, d \leftarrow 2, r \leftarrow 1, t \leftarrow 25$，而且在优先队中置 (25,10,30)。(在这个算法中，$p[i]$ = 第 i 个素数；k = 至今为止找到的素数个数；n = 素数候选者；d = 到下个候选者的距离；r = 在这个队中元素的个数；$t = p[r+2]^2$，即我们将对之增加 r 的下一个 n。队中项的形式为 $(u, v, 6p)$，其中 p 是 $u, v = 2p$ 或 $4p$ 的素因子，而且 $u+v$ 不是 3 的一个倍数。)

P2. 设 (q, q', q'') 是有最小的头一个分量的队元素。在队中以 $(q+q', q''-q', q'')$ 代替它。(这表示必须加以排除的 $q''/6$ 的下一倍数)。若 $n > q$，重复这一步骤直到 $n \leqslant q$ 为止。

P3. 如果 $n > N$，则结束本算法。否则，若 $n < q$，则置 $k \leftarrow k+1, p[k] \leftarrow n, n \leftarrow n+d, d \leftarrow 6-d$，并重复这一步骤。

P4. (现在 $n=q$ 不是质数。)如果 $n=t$，则置 $r \leftarrow r+1, u \leftarrow p[r+2], t \leftarrow u^2$，并且按照 $u \bmod 3 = 2$ 或 $u \bmod 3 = 1$ 把 $(t, 2u, 6u)$ 或 $(t, 4u, 6u)$ 插入队中。

P5. 置 $n \leftarrow n+d, d \leftarrow 6-d$，并返回步骤 P2。

于是这个计算开始如下：

队的内容	找到的质数
(25,10,30)	5,7,11,13,17,19,23
(35,20,30)(49,28,42)	29,31
(49,28,42)(55,10,30)	37,41,43,47
(55,10,30)(77,14,42)(121,22,66)	53

如果把队保持为一个堆，则我们可以在 $O(N \log N \log\log N)$ 步内求出所有 $\leqslant N$ 的素数；这个堆集的长度至多是 $<\sqrt{N}$ 的素数的个数。Eratosthenes 的筛，如同在习题 4.5.4-8 中实现的那样，是需要相当多随机存取存储的一个 $O(N \log\log N)$ 方法。7.1 节中讨论了一些更有效的实现。

16. **I1.** 置 $K \leftarrow$ 有待插入的键码；$j \leftarrow n+1$。

I2. 置 $i \leftarrow \lfloor j/2 \rfloor$。

I3. 如果 $i=0$ 或 $K_i \geqslant K$，则置 $K_j \leftarrow K$ 并且终止此算法。

I4. 置 $K_j \leftarrow K_i, j \leftarrow i$ 并返回步骤 I2。

[T. Porter 和 I. Simon 在 *IEEE Trans* SE-1(1975),292～295 中证明了，给定一致地

随机的数的堆，如果 A_{n+1} 表示步骤 4 被执行的平均次数，则对于 $n>1$，我们有 $A_n=\lfloor \lg n\rfloor+(1-n^{-1})A_{n'}$，其中 $n=(1b_{l-1}b_{l-2}\cdots b_0)_2$ 蕴涵 $n'=(1b_{l-2}\cdots b_0)_2$。如果 $l=\lfloor \lg n\rfloor$，则这个值总是 $\geqslant A_{2^{l+1}-1}=(2^{l+1}-2)/(2^{l+1}-1)$，而且总是 $\leqslant A_{2^l}<\alpha$，其中 α 是(19)中的常数。]

17. 文件 1 2 3 通过算法 H 进入堆 3 2 1，但通过习题 16 进入堆 3 1 2。(注意：后一个建立堆的方法有阶为 $N\log N$ 的最坏情况；但是经验测试已表明，在建立堆的期间，对于随机输入，步骤 2 的迭代次数大约少于 $2.28N$。R. Hayward 和 C. McDiarmid. [*J. Algorithms* **12**(1991)，126～153]已经严格地证明比例常数介于 2.2778 和 2.2994 之间)。

18. 删去步骤 H6，并以下述步骤代替 H8：

H8′. [向后移动] 置 $j\leftarrow i$，$i\leftarrow\lfloor j/2\rfloor$。

H9′. [K 合适吗？]如果 $K\leqslant K_i$ 或 $j=l$，则置 $R_i\leftarrow R$ 并返回到 H2。否则置 $R_j\leftarrow R_i$，并返回 H8′。▌

这个方法实质上与习题 16 相同，但堆具有不同的开始位置。对文件的实际修改是和算法 H 中一样的。关于这个方法的经验测试表明，在选择阶段每过筛一次，$R_j\leftarrow R_i$ 出现的次数分别以概率(.837，.135，.016)为(0，1，2)。这个方法使程序 H 稍微长些，但是把它的渐近速度改进成为 $13N\lg N+O(N)$。把一个变址寄存器的值取半的 MIX 指令在这儿很合意。

C.J.H. McDiarmid 和 B.A. Reed[*J. Algorithms* **10**(1989)，352～365]已经证明，在建立堆期间这个修改也节省平均 $(3\beta-8)N\approx0.232N$ 的比较，其中 β 在习题 27 的答案中定义。关于对 Floyd 的改进的进一步分析，请见 I. Wegener，*Theoretical Comp. Sci.* **118**(1993)，81～98。

武继刚和朱洪[*J. Comp. Sci. and Tech.* **9** (1994)，261～266]已经发现，也可使用二分查找，使得选择阶段的每个过筛至多涉及 $\lg N+\lg\lg N$ 次比较和 $\lg N$ 次移动。

19. 如同在习题 18 的修正的过筛算法那样进行，且 $K=K_N$，$l=1$ 和 $r=N-1$，并在步骤 H3 中以一个给定的 j 值开始。

20. 对于 $0\leqslant k\leqslant n$ 和对于某个 $q\geqslant0$，其二进表示为 $(b_n\cdots b_k a_1\cdots a_q)_2$ 的 $\leqslant N$ 的正整数的个数显然为 $(b_{k-1}\cdots b_0)_2+1+\sum_{0\leqslant q<k}2^q=(1b_{k-1}\cdots b_0)_2$。

21. 设 $j=(c_r\cdots c_0)_2$ 是在范围 $\lfloor N/2^{k+1}\rfloor=(b_n\cdots b_{k+1})_2<j<(b_n\cdots b_k)_2=\lfloor N/2^k\rfloor$ 中。则 s_j 是对于某个 $q\geqslant0$ 其二进表示形式为 $(c_r\cdots c_0a_1\cdots a_q)_2$ 的 $\leqslant N$ 的正整数的个数，即是 $\sum_{0\leqslant q\leqslant k}2^q=2^{k+1}-1$。因此大小为 $2^{k+1}-1$ 的非特殊的子树的个数为

$$\lfloor N/2^k\rfloor-\lfloor N/2^{k+1}\rfloor-1=\lfloor(N-2^k)/2^{k+1}\rfloor$$

[为证明这后一恒等式，以 $n=2$ 和 $x=N/2^{k+1}$ 使用习题 1.2.4-38 中的重复律。]

22. 在 $l=1$ 之前，五种可能性是 5 3 4 1 2，3 5 4 1 2，4 3 5 1 2，1 5 4 3 2 和 2 5 4 1 3。这些可能性的每一种 $a_1a_2a_3a_4a_5$ 在 $l=2$ 之前导致三种可能的排列

$a_1a_2a_3a_4a_5$，$a_1a_4a_3a_2a_5$，$a_1a_5a_3a_4a_2$。

23.(a)在 B 次迭代之后，$j\geqslant 2^Bl$；因此 $2^Bl\leqslant r$。(b) $\sum_{l=1}^{n}\lfloor\log_2(N/l)\rfloor=(\lfloor N/2\rfloor-\lfloor N/4\rfloor)+2(\lfloor N/4\rfloor-\lfloor N/8\rfloor)+3(\lfloor N/8\rfloor-\lfloor N/16\rfloor)+\cdots=\lfloor N/2\rfloor+\lfloor N/4\rfloor+\lfloor N/8\rfloor+\cdots=N-\nu(N)$，这里 $\nu(N)$ 是 N 的二进表示中 1 的个数。又由习题 1.2.4-42，我们有 $\sum_{r=1}^{N-1}\lfloor\lg r\rfloor=N\lfloor\lg N\rfloor-2^{\lfloor\lg N\rfloor+1}+2$。由定理 H 我们知道，关于 B 的这个上限在堆的建立阶段是最好的。而且注意这样一点是有趣的，就是有惟一的一个包含诸键码 $\{1,2,\cdots,N\}$ 的堆，使得在算法 H 的整个选择阶段 K 恒等于 1。(例如，当 $N=7$ 时，即堆是 7 5 6 2 4 3 1 时；不难从 N 过渡到 $N+1$。)这个堆给出对于堆排序的选择阶段 B 的极大值(连同 D 的极大值 $\lfloor N/2\rfloor-1$)，所以对于整个排序，B 的最好的上限是 $N-\nu(N)+N\lfloor\lg N\rfloor-2^{\lfloor\lg N\rfloor+1}+2$。

24. $\sum_{k=1}^{N}\lfloor\lg k\rfloor^2=(N+1-2^n)n^2+\sum_{0\leqslant k<n}k^2 2^k=(N+1)n^2-(2n-3)2^{n+1}-6$，其中 $n=\lfloor\lg N\rfloor$(参见习题 4.5.2-22)；因此最后过筛的方差是 $\beta_N=((N+1)n^2-(2n-3)2^{n+1}-6)/N-((N+1)n+2-2^{n+1})^2/N^2=O(1)$。$B'_N$ 的标准差为 $(\sum\{\beta_s\mid s\in M_N\})^{1/2}=O(\sqrt{N})$。

25. 过筛是“均匀”的，而且每个比较 $K_j:K_{j+1}$ 有 $\frac{1}{2}$ 的概率得出<。在这种情况下对于 C 的平均贡献仅仅是对于 A 和 B 的平均贡献之和的一半，即是 $\left((2n-1)2^{n-1}+\frac{1}{2}\right)\Big/(2^{n+1}-1)$。

26. (a) $\left(\frac{10}{25}+\frac{1}{2}+1\frac{3}{9}+\frac{1}{2}+1\frac{1}{2}+1\frac{2}{5}+2\frac{1}{2}+\frac{1}{2}+1\frac{1}{2}+1\frac{1}{2}+2\frac{1}{2}+1\frac{1}{2}+2+2+3+0+1+1+2+1+2+2+3+1+2+2\right)/26=1189/780\approx 1.524$。

(b) $\sum_{k=1}^{N}\nu(k)-N+\frac{1}{2}\lfloor N/2\rfloor-\frac{1}{2}n+\sum_{k=1}^{n-1}\min(\alpha_{k-1},\alpha_k-\alpha_{k-1}-1)/(\alpha_k-1))/N$，这里 $\nu(k)$ 是在 k 的二进表示中二进位 1 的个数，且 $\alpha_k=(1b_k\cdots b_0)_2$。如果 $N=2^{e_1}+2^{e_2}+\cdots+2^{e_t}$，且 $e_1>e_2>\cdots>e_t\geqslant 0$，则可以证明 $\sum_{k=0}^{N}\nu(k)=\frac{1}{2}((e_1+2)2^{e_1}+(e_2+4)2^{e_2}+\cdots+(e_t+2t)2^{e_t})+t-N$。[借助于 Mellin 转换可以清楚地分析这样一些和的渐近性质。参见 Flaiolet，Grabner，Kirschenhofer，Prodinger 和 Tichy，*Theoretical Comp. Sci.* **123**(1994)，291～314]。

27. J. W. Wrench Jr. 已经发现，一般的 Lambert 级数 $\sum_{n\geqslant 1}a_nx^n/(1-x^n)$ 可展开成为 $\sum_{N\geqslant 1}(\sum_{d\backslash N}a_d)x^N=\sum_{m\geqslant 1}(a_m+\sum_{k\geqslant 1}(a_m+a_{m+k})x^{km})x^{m^2}$。

[$a_n=1$ 和 $a_n=n$ 的情况是由 J. H. Lambert 在他的 *Anlage zur Architectonic*, **2** (里加，1771)，§875 中引进的；Clausen 在 *Crelle* **3**(1828)，95 中对于 $a_n=1$ 的情况指出了他的公式。还有 H. F. Scherk 在 *Crelle* **9**(1832)，162～163 中给出一个证明。当 $a_n=n$ 和 $x=\frac{1}{2}$ 时，我们得到关系

$$\beta = \sum_{n\geqslant 1}\frac{n}{2^n-1} = \sum_{m\geqslant 1}\left(m\left(\frac{2^m+1}{2^m-1}\right)+\frac{2^m}{(2^m-1)^2}\right)2^{-m^2} =$$

$$2.74403\ 38887\ 59488\ 36048\ 02148\ 91492\ 27216\ 43114+$$

这个常数出现于(20)中,其中我们有 $B'_N \sim (\beta-2)N$ 和 $C'_N \sim \left(\frac{1}{2}\beta-\frac{1}{4}\alpha-\frac{1}{2}\right)N$]。

附带说一下,如果在习题 5.1.1-16 的头一个恒等式中我们置 $q=x$ 和 $z=xy$,就可得到有趣的恒等式

$$\sum_{n\geqslant 1}\frac{x^n}{1-x^n} = \sum_{k\geqslant 1}kx^k(1-x^{k+1})(1-x^{k+2})\cdots$$

28. 节点 k 的儿子是节点 $3k-1, 3k$ 和 $3k+1$;父节点是$\lfloor(k+1)/3\rfloor$。类似于程序 H 的一个 MIX 程序花费近似于 $21\frac{2}{3}\log N \approx 13.7\lg N$ 个时间单位。利用习题 18 的思想,这个数降低成为 $18\frac{2}{3}N\log_3 N \approx 11.8N\lg N$,尽管除以 3 将增加一个很大的 $\Theta(N)$项。

有关 t 叉堆进一步的信息,请见 S. Okoma, *Lecture Notes in Comp. Sci.* **88** (1980), 439~451。

30. 假设 $n=2^t-1+r$,其中 $t=\lfloor\lg n\rfloor$且 $1\leqslant r\leqslant 2^t$。于是 $h_{2m}=[m=0]$和通过考虑在 K_1 的位置处过筛之后可以最后在 K_{n+1}的位置栖息的级 j 上的元素个数,有对于$n\geqslant 2$

$$h_{(n+1)m} \leqslant \sum_{j=0}^{t-2}(2^j-1)h_{n(m-j)} + 2^{t-1}h_{n(m-t+1)} + rh_{n(m-t)}$$

因此如果 $g_{nm}=h_{nm}/2^m$,我们有

$$g_{(n+1)m} \leqslant \sum_{j=0}^{t-2}\frac{2^j-1}{2j}g_{n(m-j)} + g_{n(m-t+1)} + \frac{r}{2^t}g_{n(m-t)} \leqslant (\lg(n+1))\max_{m\geqslant 0}g_{nm}$$

而且由此通过归纳法得出 $g_{nm}\leqslant L_n=\prod_{k=2}^{n}\lg k$。

在选择阶段期间平均提升的总数是 $B''_N=h_N^{-1}\sum_{m\geqslant 0}mh_{Nm}$,其中 $h_N=\sum_{m\geqslant 0}h_{Nm}$是可能的堆的总数(定理 H)。我们知道 $B''_N\leqslant N\lceil\lg N\rceil$。另一方面,我们有 $B''_N\geqslant m-h_N^{-1}\sum_{k=1}^{m}(m-k)h_{Nk}\geqslant m-h_N^{-1}L_N\sum_{k=1}^{m}(m-k)2^k>m-2^{m+1}h_N^{-1}L_N$,对所有的 m。选择 $m=\lg(h_N/L_N)+O(1)$现在给出 $B''_N\geqslant\lg(h_N/L_N)+O(1)$。

由习题 23(b),为建立一个堆所需比较次数至多是 $2N$;因此 $h_N\geqslant N!/2^{2N}$。显然 $L_N\leqslant(\lg N)^N$,所以我们有 $\lg(h_N/L_N)\geqslant N\lg N-N\lg\lg N+O(N)$。[*J. Algorithms* **15**(1993), 76~100。]

31. (J. Edighoffer 给出的解,1981) 设 A 是使得对于 $1<i\leqslant n$, $A[2\lfloor i/2\rfloor]\leqslant A[2i]$和 $A[2\lfloor i/2\rfloor-1]\geqslant A[2i-1]$的 $2n$ 个元素的数组;其次我们要求对于 $1\leqslant i\leqslant n$, $A[2i-1]\geqslant A[2i]$。(对于所有的 i,这后一条件成立的充要条件是,它对于

$n/2 < i \leqslant n$ 成立,这是由于堆的结构所致)。这"孪生"的堆包含 $2n$ 个元素;为处理奇数个数的元素,我们只需甩开一个元素到边上。适当修改本小节中其它算法,可以用来保持孪生的堆,推出这些细节是有趣的。这一思想由 J. van Leeuwen 和 D. Wood 独立地发现并作了进一步的发展[*Comp. J.* **36**(1993),209~216],他们把这个结构称做"区间堆"。

32. 在 N 个元素的任何堆中,最大的 $m = \lceil N/2 \rceil$ 个元素形成一个子树。其中至少有 $\lfloor m/2 \rfloor$ 个元素必不是该子树的叶,因为具有 k 个叶的一个二叉树至少有 $k-1$ 非叶。因此最大的 m 个元素的至少 $\lfloor m/2 \rfloor$ 个出现在堆的头 $\lfloor N/2 \rfloor$ 个位置中。这些元素在达到它们最后的目的地之前必然被提升到根位置处;所以由习题 1.2.4-42,它们的移动至少对 B 贡献 $\sum_{k=1}^{\lfloor m/2 \rfloor} \lfloor \lg k \rfloor = \frac{1}{2} m \lg m + O(m)$。因此 $B_{\min}(N) \geqslant \frac{1}{4} N \lg N + O(N) + B_{\min}(\lfloor N/2 \rfloor)$,这一结果通过对 N 的归纳法得出。[I. Wegener, *Theoretical Comp. Sci* **118**(1993),81~98,定理 5.1。Schaffer 和 Sedgewick,还有 Bollobas、Fenner 和 Frieze 独立地构造了要求不多于 $\frac{1}{2} N \lg N + O(N \log \log N)$ 个提升的排列;参见 *J. Algorithms* **15**(1993),76~100;**20**(1996),205~217。根据习题 30 的结果,这样的排列是十分稀少的。]

33. 设 P、Q 指向给定的优先队;如同在正文中一样,下列算法使用约定 DIST(Λ) = 0,尽管 Λ 实际上不是一个节点。

M1. [初始化] 置 R←Λ。

M2. [表合并] 如果 Q = Λ,则置 D←DIST(P)并转到 M3。如果 P = Λ 则置 P←Q,D←DIST(P)并转到 M3。否则如果 KEY(P)≥KEY(Q),则置 T←RIGHT(P),RIGHT(P)←R,R←P,P←T 并重复步骤 M2。如果 KEY(P)<KEY(Q),则置 T←RIGHT(Q),RIGHT(Q)←R,R←Q,Q←T 并重复步骤 M2。(这一步骤实质上合并给定树的两个"右表",它暂时把向上的指针插入到诸 RIGHT 字段中。)

M3. [完成?] 如果 R = Λ 则结束此算法;P 指向答案。

M4. [修正诸 DIST] 置 Q←RIGHT(R)。如果 DIST(LEFT(R))<D,则置 D←DIST(LEFT(R)) + 1,RIGHT(R)←LEFT(R),LEFT(R)←P;否则置 D←D + 1,RIGHT(R)←P。最后置 DIST(R)←D,P←R,R←Q,并返回 M3。▌

34. 对于整个生成函数 $L(z) = \sum_{n \geqslant 0} l_n z^n = \sum_{m \geqslant 1} L_m(z)$ 的诸部分,以递推式

$$L_1(z) = z, \quad L_{m+1}(z) = L_m(z)\Big(L(z) - \sum_{k=1}^{m-1} L_k(z)\Big)$$

开始。其中 $L_m(z) = z^{2^{m-1}} + \cdots$ 生成从根到 Λ 具有最短长度 m 的左倾树,Rainer Kemp 已经证明 $L(z) = z + \frac{1}{2} L(z)^2 + \frac{1}{2} \sum_{m \geqslant 1} L_m(z)^2$,而且 $a \approx 0.25036$ 和 $b \approx 2.7494879$。[*Inf. Proc. Letters* **25**(1987),227~232;*Random Graphs* **87**(1990),103~130]。Luis Trabb Pardo 在 1978 年注意到生成函数 $G(z) = zL(z)$ 满足漂亮的关

系 $G(z)=z+G(zG(z))$。

35. 设删去的节点的 DIST 字段是 d_0，并设合并后的子树的 DIST 字段是 d_1。如果 $d_0=d_1$，则我们根本不必向上走。如果 $d_0>d_1$ 则 $d_1=d_0-1$；而且如果我们向上 n 级，则 P 的诸祖宗的诸新 DIST 字段必须分别是 $d_1+1, d_1+2, \cdots, d_1+n$。如果 $d_0<d_1$，则向上的通路只能向左拐。

36. 不用一般的优先队，最简单的是使用一个双重链接表；每当使用诸节点时，就把这些节点移向表的一端，并从另一端删去节点[见 6.1 节中关于"自组织文件"的讨论]。

37. 在一个无穷堆中，第 k 个最大的元素同等可能地出现在它的更大的祖先的左或右子堆中。因此我们可以使用数字查找树理论，在等式 6.3-(13)的记号下，得到 $e(k)=\bar{C}_k-\bar{C}_{k-1}$通过练习 6.3-28 我们得到 $e(k)=\lg k+\gamma/(\ln 2)+\frac{1}{2}-\alpha+\delta_0(k)+O(k^{-1})\approx\lg k-.274$，其中 α 在(19)中定义，而 $\delta_0(k)$是 $\lg k$ 的一个周期函数[P. V. Poblete, *BIT* **33**(1993), 411～412]。

38. $M_0=\varnothing$；$M_1=\{1\}$；对于 $N>1$，$M_N=\{N\}\uplus M_{2^k-1}\uplus M_{N-2^k}$，其中 $k=\lfloor\lg(2N/3)\rfloor$。

5.2.4 小节

1. 以 $i_1=\cdots=i_k=1, j=1$ 开始。重复地找 $\min(x_{1i_1},\cdots,x_{ki_k})=x_{ri_r}$，并置 $z_j\leftarrow x_{ri_r}, j\leftarrow j+1, i_r\leftarrow i_{r+1}$。(在这种情况下，$x_{i(m_i+1)}=\infty$ 的使用是一项决定性的妙着。)

当 k 相当大时，最好是像在 5.2.3 小节中所讨论的那样，在适合于重复选择的一株树结构中保持键码 $x_{1i_1},\cdots,x_{ki_k}$，使得在第一次之后每次找到极小值仅仅需要作$\lfloor\lg k\rfloor$次比较。其实，这是在一个优先队中"最小者先出"原理的一个典型应用。诸键码可以保持作一个堆，而且可以完全避免∞。进一步的讨论请看 5.4.1 小节。

2. 设 C 是比较的次数；我们有 $C=m+n-S$，其中 S 是在步骤 M4 或 M6 中传送的元素个数。容易看出对于 $1\leqslant s\leqslant m+n$，$S\geqslant s$ 的概率是

$$q_s=\left(\binom{m+n-s}{m}+\binom{m+n-s}{n}\right)\bigg/\binom{m+n}{n}$$

对于 $s>m+n$，$q_s=0$。因此 S 的均值是 $\mu_{mn}=q_1+q_2+\cdots=m/(n+1)+n/(m+1)$[参见习题 3.4.2-5,6]，而且方差是 $\sigma_{mn}^2=(q_1+3q_2+5q_3+\cdots)-\mu_{mn}^2=m(2m+n)/(n+1)(n+2)+(m+2n)n/(m+1)(m+2)-\mu_{mn}^2$。于是

$$C=(\min \min(m,n), \text{ave } m+n-\mu_{mn}, \max m+n-1, \text{dev } \sigma_{mn})$$

当 $m=n$ 时，这个平均值首先由 H. Nagler 计算出来，*CACM* **3**(1960), 618～

620;它渐近于 $2n-2+O(n^{-1})$,并有 $\sqrt{2}+O(n^{-1})$ 的标准离差。于是 C 徘徊地接近于它的极大值。

3. **M2′.** 如果 $K_i<K_j'$ 则转到 M3′;如果 $K_i=K_j'$,则转到 M7′;如果 $K_i>K_j'$,则转到 M5′。

M7′. 置 $K_k''\leftarrow K_j'$, $k\leftarrow k+1$, $i\leftarrow i+1$, $j\leftarrow j+1$。如果 $i>M$,转到 M4′;否则如果 $j>N$ 转到 M6′;否则返回 M2′。 ▌

(对于算法 M 的其它步骤也作了适当的修改。此外,如果我们在这个文件的末尾插入人为的键码 $K_{M+1}=K_{N+1}'=\infty$,则许多特殊情况就会消失。)

4. 随着时间的推移,出现在选择树的一个固定的内节点处的元素序列,是通过合并出现于该节点的儿子处的元素序列得到的(5.2.3 节中的这个讨论是以选择最大元素为基础的,但是它同样也可以应用于颠倒次序的情况)。所以在树选择中进行的操作,实质上和合并中进行的那些操作相同,但它们以不同的序列而且使用不同的数据结构来实施。

习题 1 中指出了合并和树选择之间的另一个关系。注意,一些单元素文件的 N 路合并是一个选择排序;请把 (A,B,C,D) 的四路合并和先是 (A,B),(C,D) 然后 (AB,CD) 的两路合并作比较。

5. 在步骤 N6 中,我们总有 $K_i<K_{i-1}\leqslant K_j$;在 N10 中,$K_j<K_{j+1}<K_i$。

6. 2 6 4 10 8 14 12 16 15 11 13 7 9 3 5 1。在一次扫描之后我们有 1 2 5 6 7 8 13 14 16 15 12 11 10 9 4 3,预期的两个下坡消失。这种可能性是由 D. A. Bell 说明的,*Comp. J.* **1**(1958),74。类似于此的怪例,使我们几乎没有希望来对于算法 N 作一个精细的分析。

7. 如果 $N>1$,则为 $\lceil \lg N\rceil$。(考虑必须把 p 加倍多少次,直到它 $\geqslant N$ 为止)。

8. 如果 N 不是 $2p$ 的一个倍数,则在这趟扫描中有一个短路段,而且它总是接近于中间:设它的长度是 t,我们有 $0\leqslant t<p$,步骤 S12 处理的是短路段和空路段"合并",或 $t=0$ 的情况;否则,我们实质上有 $x_1\leqslant x_2\leqslant\cdots\leqslant x_p\,|\,y_t\geqslant\cdots\geqslant y_1$。如果 $x_p\leqslant y_t$ 则左边的路段首先穷尽,而且传送了 x_p 之后,步骤 S6 将使我们达到步骤 S13。另一方面,如果 $x_p>y_t$,则右边将被人为地穷尽,但在步骤 S3 中 $K_j=x_p$ 将决不 $<K_i$! 于是,在所有的情况下,S6 将最终地使我们到达 S13。

10. 例如,如果 $j\geqslant n$,则算法可以把方法 $x_{j+1}\cdots x_{j+m}$ 同 $x_{j+m+1}\cdots x_{j-m+n}$ 无冲突地合并到一个阵列的 $x_1\cdots x_{m+n}$ 诸位置上。留点心我们就可以进一步利用这个思想,使得整个排序仅需要 $N+2^{\lfloor \lg N\rfloor -1}$ 个单元。但是这个程序相对于算法 S 来说稍微复杂些[*Comp. J.* **1** (1958),75;也见 L. S. Lozinskii,*Kibornetika* **1**,**3**(1965),58~62]。

11. 是的。例如,这可以通过考虑与习题 4 提出的树选择的关系看出。但显然算法 N 和 S 不是稳定的。

12. 置 $L_0\leftarrow 1$, $t\leftarrow N+1$;然后对于 $p=1,2,\cdots,N-1$ 作下列工作:

如果 $K_p \leqslant K_{p+1}$，则置 $L_p \leftarrow p+1$；否则置 $L_t \leftarrow -(p+1)$，$t \leftarrow p$。最后置 $L_t \leftarrow 0$，$L_N \leftarrow 0$，$L_{N+1} \leftarrow |L_{N+1}|$。

（保持了稳定性。扫描次数是$\lceil \lg r \rceil$，其中 r 是输入中递增路段的个数；5.1.3 节中分析了 r 的精确分布。我们可以得出结论：当使用链接分配时，自然合并比直接合并更可取，但使用顺序分配时它是低劣的。）

13. 对于 $N \geqslant 3$ 的运行时间是$(11A+6B+3B'+9C+2C''+4D+5N+9)u$，其中 A 是扫描次数；$B=B'+B''$是所实施的子文件合并操作的次数，这里 B'是其中 p 子文件首先被穷尽的合并次数；$C=C'+C''$是所实施的比较次数，其中 C'是满足 $K_p \leqslant K_q$ 的比较的次数；$D=D'+D''$是当另一个子文件已被穷尽时在诸子文件中剩下的元素个数，其中 D'是属于 q 子文件的这样元素的个数。在表 3 中，我们有 $A=4$，$B'=6$，$B''=9$，$C'=22$，$C''=22$，$D'=10$，$D''=10$。总共的时间 $=761u$。（当像在习题 5.2.1-33 中那样进行改进后，可比较的程序 5.2.1L 只花费 $433u$，所以我们看出当 N 很小时，合并并不特别有效）。

算法 L 对诸子文件进行一系列的合并，这些子文件的大小(m,n)，可如下确定：设在二进制记号下 $N-1=(b_k\cdots b_1b_0)_2$。对于$(m,n)=(2^j,2^j)$，$0\leqslant j\leqslant k$，有$(b_k\cdots b_{j+1})_2$ 个“通常的”合并；而且对于 $0\leqslant j\leqslant k$，每当 $b_j=1$ 时，对于$(m,n)=(2^j,1+(b_{j-1}\cdots b_0)_2)$有“特殊的”合并。例如当 $N=14$ 时，有六个通常的(1,1)合并，三个通常的(2,2)合并，一个通常的(4,4)合并，以及处理大小为(1,1)，(4,2)，(8,6)的子文件的特殊的合并。合并大小为(m,n)的多重集合 M_N 也可通过下列递推关系来加以描述：

$$M_1=\emptyset;\quad M_{2^k+r}=\{(2^k,r)\}\uplus M_{2^k}\uplus M_r\quad \text{对于 } 0<r\leqslant 2^k$$

由此得出，不论输入分布如何，我们有 $A=\lceil \lg N \rceil$，$B=N-1$，$C'+D''=\sum_{j=0}^{k} b_j 2^j\left(1+\frac{1}{2}j\right)$，$C''+D'=\sum_{j=0}^{k} b_j\left(1+2^j\left(\frac{1}{2}j+b_{j+1}+\cdots+b_k\right)\right)$；因此仅仅 B'，C'，D'需要进一步分析。

如果对于算法 L 的输入是随机的，则每个合并操作都满足习题 2 的条件，而且同其它合并的特性无关；所以 B'，C'，D'的分布是每个子文件合并时它们各自的分布的卷积。这样的合并的平均值是 $B'=n/(m+n)$，$C'=mn/(n+1)$，$D'=n/(m+1)$。对于所有有关的(m,n)把这些加起来，就得精确的平均值。

当然，当 $N=2^k$ 时，我们有最简单的情况：$B'_{\text{ave}}=\frac{1}{2}B$，$C'_{\text{ave}}=\frac{1}{2}C_{\text{ave}}$，$C+D=kN$，而且 $D_{\text{ave}}=\sum_{j=1}^{k}(2^{k-j}2^j/(2^{j-1}+1))=\alpha' N+O(1)$，其中 $\alpha'=\sum_{n\geqslant 0}\frac{1}{2^n+1}=\alpha+\frac{1}{2}-2\sum_{n\geqslant 1}\frac{1}{4^n-1}=1.26449978034844420919131974725549848255 77$—可以像在习题 5.2.3-27 中那样计算到很高的精确度。这个特殊情况是由 A. Gleason[未发表，1956]和 H. Nagler[*CACM* **3**(1960)，618～620]首先分析的。

14. 在习题 13 中置 $D=B$ 以使 C 取得极大值。[W. Panny 和 H. Prodinger, *Algorithmica* **14**(1995), 340～354 已经对算法 L 进行了详细的分析]。

15. 对于已知 L_s 等于 p 或 q 的诸情况，作步骤 L3, L4, L6 的额外的副本。[只需简单地更改寄存器的名字，也可作进一步的改进，并且删除内循环中的赋值 $s \leftarrow p$（或 $s \leftarrow q$）！例如把行 20 和行 21 改变成为“LD3 INPUT,1(L)”而且以 rI3 之值为 p，rI1 之值为 s，以及 L_s 等于 p 来继续运行。对应于 (p, q, r) 相对于(rI1, rI2, rI3)的不同排列，以及根据 L_s 的不同情况，我们可以构作内循环的 12 个副本，从而把平均运行时间减少到 $8N\lg N + O(N)$]。

16.（这个结果将比算法 L 稍快；参见习题 5.2.3-28。）

17. 把新的记录当作长度为 1 的子文件。如果最小的两个子文件有相同长度，即重复地合并之。（得到的排序算法实质上和算法 L 相同，但在不同的相对时间合并诸子文件。）

18. 是的，但它似乎是一项复杂的工件。要找出的头一个解作用下列巧妙的构造。[*Doklady Akad Nauk SSSR* **186**(1969), 1256～1258]：设 $n \approx \sqrt{N}$。把这个文件分成为 $m+2$ 个“段” $Z_1 \cdots Z_m Z_{m+1} Z_{m+2}$，其中 Z_{m+2} 包含 $(N \bmod n)$ 个记录，而每一个其它的段恰包含 n 个记录。把 Z_{m+1} 的诸记录同包含 R_M 的段进行交换；现在这个文件有 $Z_1 \cdots Z_m A$ 的形式，其中 $Z_1 \cdots Z_m$ 中的每一个恰巧包含 n 个排好序的记录，而且这里 A 是包含 s 个记录的一个辅助区域，其中的 s 在范围 $n \leqslant s < 2n$ 中。

找出具有最小前导元素的段，而且把整个该段同 Z_1 作交换；如果有一个以上的段有最小前导元素，则选择有最小尾元素的一个段。（这花费 $O(m+n)$ 个操作。）然后找出具有次小前导元素和尾元素的段，而且把它同 Z_2 进行交换，等等。最后，通过 $O(m(m+n)) = O(N)$ 个操作，我们重新安排了 m 个段，使得它们的前导元素是有序的。而且，由于对于该文件原来的假定，在 $Z_1 \cdots Z_m$ 中的每个键码现在都有少于 n 个反序。

我们利用下列技巧，可以合并 Z_1 和 Z_2：把 Z_1 同 A 的头 n 个元素 A' 进行交换；然后以通常方式合并 Z_2 和 A'，但当它们被输出时与 $Z_1 Z_2$ 的元素相交换。例如，如果 $n=3$，且 $x_1 < y_1 < x_2 < y_2 < x_3 < y_3$，则我们有

	段 1	段 2	辅助区域
初始值的内容：	$x_1x_2x_3$	$y_1y_2y_3$	$a_1a_2a_3$
交换 Z_1：	$a_1a_2a_3$	$y_1y_2y_3$	$x_1x_2x_3$
交换 x_1：	$x_1a_2a_3$	$y_1y_2y_3$	$a_1x_2x_3$
交换 y_1：	$x_1y_1a_3$	$a_2y_2y_3$	$a_1x_2x_3$
交换 x_2：	$x_1y_1x_2$	$a_2y_2y_3$	$a_1a_3x_3$
交换 y_2：	$x_1y_1x_2$	$y_2a_2y_3$	$a_1a_3x_3$
交换 x_3：	$x_1y_1x_2$	$y_2x_3y_3$	$a_1a_3a_2$

（当辅助区域的第 n 个元素被交换过后，这个合并就总是完成了；这个方法一般会打

乱辅助记录的原来位置。)

上述技巧用来合并 Z_1 和 Z_2,然后 Z_2 和 Z_3,…,Z_{m-1}和 Z_m,总共需要 $O(mn) = O(N)$个操作。由于没有多于 n 个反序的元素,这个文件的 $Z_1 \cdots Z_m$ 部分已经完全被排好序。

为了最后的"清理",我们通过在 $O(s^2) = O(N)$步内进行插入来对 $R_{N+1-2s} \cdots R_N$ 排序;这把 s 个最大元素带到区域 A 中。然后利用上述技巧,以及辅助存储区域 A(但整个地交换右和左,小于和大于的作用),合并 $R_1 \cdots R_{N-2s}$和 $R_{N+1-2s} \cdots R_{N-s}$。最后,我们通过插入来对 $R_{N+1-s} \cdots R_N$ 排序。

J. Katajainen, T. Pasanen 和 J. Teuhola 在*Nordic J. Computing* **3**(1996),27～40 中讨论了随后的改进。关于现场稳定合并的问题,请见 5.5-3 的答案。

19. 我们可以把输入的车辆编号,使得它们最后的排列依次为 1　2…2^n;所以这实质上是一个排序问题。首先把 2^{n-1}辆车移过 $n-1$ 个栈,以递减的次序来放置它们。然后把它们转移到第 n 个栈,使得最小的在顶上。然后使其它 2^{n-1}辆车通过 $n-1$ 个栈,以递增的顺序放置它们,并使它们停在第 n 个栈之前。最后以显然的形式把两个序列合并在一起。

20. 关于进一步的信息,请见 R. E. Tarjan, *JACM* **19**(1972),341～346。

22. 参见*Information Processing Letters* **2**(1973),127～128。

23. 这些合并可以通过二叉树表示,这二叉树的所有外节点在级$\lfloor \lg N \rfloor$和$\lceil \lg N \rceil$上。因此比较的极大次数是有 N 个外节点的一棵二叉树的极小外路径长度,即等式 5.3.1-(34)减去 $N-1$,因为 $f(m,n) = m+n-1$ 给出极大值,而且有 $N-1$ 次合并。(也参见等式 5.4.9-(1)。)

P. Flaiolet 和 M. Golin 在*Acta Informatica* **31** (1994),679～696 已经给出借助于 Mellin 转换来研究这样的递推式的渐近性质的通用技术;特别是,他们证明,比较的平均次数是 $N \lg N - \theta N + \delta(\lg N)N + O(1)$,方差是$\approx .345N$,其中 δ 是周期为 1 和均值为 0 的一个连续函数,而且

$$\theta = \frac{1}{\ln 2} - \frac{1}{2} + \frac{1}{\ln 2}\sum_{m=1}^{\infty} \frac{2}{(m+1)(m+2)} \ln \frac{2m+1}{2m} =$$

$$1.24815\ 20420\ 99653\ 84890\ 29565\ 64329\ 53240\ 16127 +$$

当 $N \to \infty$ 时通过一个正态分布可以很好地逼近比较的总数;参见黄显贵和 M. Cramer 在*Random Structures and Algorithms* **8** (1996),319～336; **11**(1997),81～96 上的补充分析。

5.2.5 小节

1. 否。因为基数排序全然无效,除非在头一次扫描之后,分布排序是稳定的。(但如同正文最后一段中所提议的那样,所提出的分布排序可用于首先最高位数字的基数排序方法中,并且推广基数交换。)

2. 恰恰相反,它是"反稳定的";具有相同键码的元素以相反的次序出现,因为头

一遍扫描从 R_N 到 R_1 遍历诸记录。(这证明是方便的,因为程序 R 的行 28 和行 20 把 Λ 同 0 等置起来;但是当然没有必要向后进行头一次扫描。)

3. 如果堆 0 非空,则 BOTM[0]已经指向头一个元素;如果它是空的,则我们置 P ←LOC(BOTM[0])而且过后使 LINK(P)指向头一个非空堆的底。

4. 当剩下偶数次扫描时,首先取堆 0(由顶到底),接着取堆 1,…,堆($M-1$);这个结果相对于至今所考虑的数字来说将是有序的。当剩下奇数次扫描时,首先取第 $M-1$堆,然后取第 $M-2$ 堆,…,第 0 堆;这个结果相对于至今所考察的数字来说将是逆序的。(这个规则显然是由 E. H. Friend 首先提出的[*JACM* **3**(1956),156,165～166]。)

5. 把行 04 改变成为“ENT3 7”并把表 R3SW 和 R5SW 改变成为

```
R3SW     LD2     KEY,1(1:1)
         LD2     KEY,1(2:2)
         LD2     KEY,1(3:3)
         LD2     KEY,1(4:4)
         LD2     KEY,1(5:5)
         LD2     INPUT,1(1:1)
         LD2     INPUT,1(2:2)
         LD2     INPUT,1(3:3)
R5SW     LD1     INPUT,1(LINK)
          ⋮      (再重复上一行 6 次)
         DEC1    1
```

通过在每处把“3”改成“8”,可以求得新的运行时间;对于 $p=8$,它总计为$(11p\text{-}1)N+16pM+12p-4E+2$。

6. (a)考虑放置第 $N+1$ 个元素。递推式

$$p_{M(N+1)k} = \frac{k+1}{M}p_{MN(k+1)} + \frac{M-k}{M}p_{MNk}$$

等价于所述的公式。(b)通过对 n 用归纳法可证第 n 次导数满足 $g_{M(N+1)}^{(n)}(z)=(1-n/M)g_{MN}^{(n)}(z)+((1-z)/M)g_{MN}^{(n+1)}(z)$。置 $z=1$,我们求出 $g_{MN}^{(n)}(1)=(1-n/M)^N M^{\underline{n}}$,因为 $g_{M0}(z)=z^M$。因此 $\text{mean}(g_{MN})=(1-1/M)^N M$, $\text{var}(g_{MN})=(1-2/M)^N M(M-1)+(1-1/M)^N M-(1-1/M)^{2N}M^2$。

(注意,程序 R 中 E 的生成函数是 $g_{MN}(z)^p$。)

7. 设 R=基数排序,RX=基数交换。某些重要的类似性和差别是:RX 从最高位数字进行到最低位,而 R 则从另一条途径进行。这两个方法都通过数字检查来进行排序,而不作键码的比较。RX 总有 $M=2$(但见习题 1)。R 的运行时间几乎总是不变的,而 RX 对于数字的分布是敏感的。在两种情况下运行的时间都是 $O(N\log K)$,其中 K 是键码的范围,但 RX 的比例常数较高;另一方面,当键码按它们的前导数字来说是均匀分布时,无论 K 的大小如何,RX 总有 $O(N\log N)$的平均运行时间。R 需要链接字段而 RX 在“极小的存储”下运行。R 的内循环更适合于“流水线”计算机。

8. 在最后的扫描中,诸堆应以另一个次序被钩在一起;例如,如果 $M=256$,则堆$(10000000)_2$ 首先来到,然后堆$(10000001)_2$…,堆$(11111111)_2$,堆$(00000000)_2$,堆$(00000001)_2$,…,堆$(01111111)_2$。通过修改算法 H,或(在表 1 中)通过在最后一

次扫描时改变存储分配策略，即可更容易地进行挂钩顺序的改变。

9.如同在习题 5.2.2-33 中那样，我们可以首先把负的键码同正的键码分开；或者我们可以在头一次扫描时把键码改成补码记号。或者，在最后一次扫描之后，可以把正键码同负键码分开，颠倒后者的顺序，但习题 5.2.2-33 的方法已不再能用了。

11.若没有头一次扫描，这个方法仍将正确地进行排序，因为（碰巧）503 已在 509 之前。若没有头两次扫描，反序数将是 $1+1+0+0+0+1+1+1+0+0=5$。

12.在步骤 M4（习题 5.2-12）中交换 R_k 同 $R[P]$ 之后，我们可以比较 K_k 同 K_{k-1}。如果 K_k 较小，则把它同 $K_{k-2}, K_{k-3}, \cdots$，作比较，直到求出 $K_k \geqslant K_j$ 为止。然后置 $(R_{j+1}, \cdots, R_{k-1}, R_k) \leftarrow (R_k, R_{j+1}, \cdots, R_{k-1})$，而不改变 LINK 字段。设置一个人为的键码 K_0 很方便，它≤在这个文件左边的所有其它键码。

14.如果在习题 5.1.3-20 的意义下，卡片原来的排列需要读 k 次，而且如果对于每次扫描我们使用 m 个堆，则必须至少进行 $\lceil \log_m k \rceil$ 次扫描。（考虑从一个已排好序的卡片组到原来的卡片组的逆过程；每次扫描至多使读的次数增加 m 的一个因子。）给定的排列需要 4 次递增的读，10 次递减的读；所以递减的次序需要具有两个堆的 4 次扫描或者具有三个堆的 3 次扫描。

反之，这个最优的扫描次数可被达到；根据卡片是在第几次被读的，重新把它们编号成为 0 到 $k-1$，并且使用一个基数排序（在基数 m 下最低位数字先开头）。[参见 Martin Gardner 的*Sixth Book of Mathematical Games*（三藩市：W. H. Freeman，1971），111～112。]

15.设有 k 次读和 m 个堆。每次扫描时次序都被颠倒；如果按一个次序读 k 次，则在相反的次序下读的次数为 $n+1-k$。极小扫描次数或者是大于等于 $\log_m k$ 的最小偶数，或者是大于等于 $\log_m(n+1-k)$ 的最小奇数。（反向进行时，在一次扫描之后顶多有 m 次递减的读，在两次扫描之后，顶多有 m^2 次递增的读，等等。）这个例子可在 $\min(2,5)=2$ 次扫描中被排成递增次序，在 $\min(3,4)=3$ 次扫描中被排成递减次序，同时仅使用两个堆。

16.假定每个串之后跟着一个特殊的空字符，它小于字母表的任何字母。由在一个数据块区中链接在一起的所有串开始，实施一个具体左至右的基数排序。然后对于 $k=1,2\cdots$ 修改包含一个以上不同串的每个块区，办法是根据每个串的第 k 个字母，把它分成为一些子块区，同时按照它们已被考察的前缀，使这些块区保持有序。当一个块区仅有一个项时，或者当它的第 k 个字符为全空（因而它的键码相等），我们可以安排避免再对它进行考察。[R. Paige 和 R. E. Tarjan，*SICOMP* **16**（1987），973～989，§2。]这个过程实质上和在 6.3 节中构造一个检索结构是一样的。Aho，Hopcroft 和 Ullman 的*The Design and Analysis of Computer Algorithms*（Addison-Wesley，1974），79～84 对这个问题给出了以从右到左的基数排序为基础的一个更简单但不大有效的算法。正文中所引的 McIlroy，Bostic 和 McIlroy 的方法，更实用和更快。

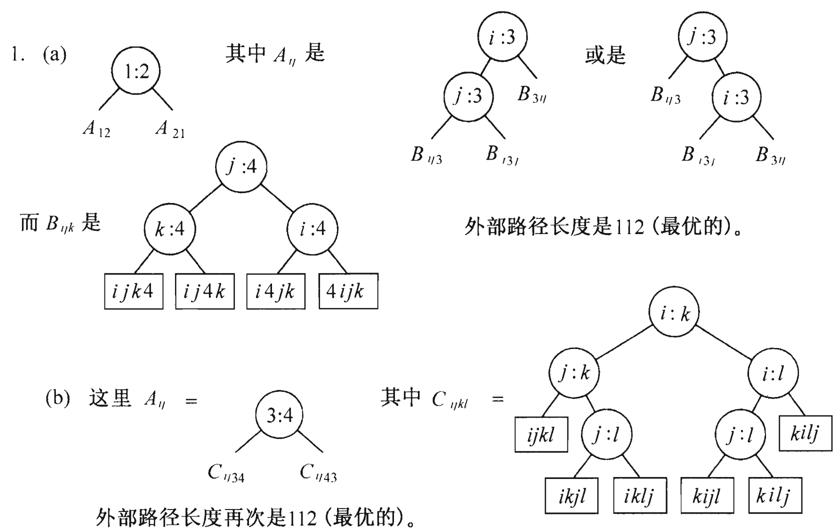
17.MacLaren 的方法加快了第二级，但在顶层一级不能使用它，因为它不能计算数 N_k。

18. 首先我们证明提示：设 $p_k=\int_{k/CN}^{(k+1)/CN}f(x)\mathrm{d}x$ 是当有 CN 个箱时一个键码落入箱 k 的概率。为分布诸记录所需时间是 $O(N)$，而在分布之后剩下的反序的平均个数为 $\frac{1}{2}\sum_{k=0}^{CN-1}\bar{Z}_j\binom{N}{j}p_k^j(1-p_k)^{N-j}\binom{j}{2}=\frac{1}{2}\sum_{k=0}^{cN-1}\binom{N}{2}p_k^2\leqslant\frac{N-1}{4}\sum_{k=0}^{cN-1}p_kB/C$，因为 $p_k\leqslant B/CN$。

现在考虑两级分布，且有 cN 个顶层一级的箱子，并设 $b_k=\sup\{f(x)\mid k/cN\leqslant x<(k+1)/cN\}$。于是平均的总共运行时间为 $O(N)$ 加上 $\sum_{k=0}^{cN-1}T_k$，其中 T_k 是 MacLaren 方法对于有密度函数 $f_k(x)=f((k+x)/cN)/cNp_k$ 的 N_k 个键码进行排序所需要的平均时间。通过这个提示，我们有 $T_k=\mathrm{E}\ O(b_kN_k/cNp_k)$ 因为 $f_k(x)$ 是以 b_k/cN_{p_k} 为限的。但 $\mathrm{E}N_k=Np_k$，所以 $T_k=O(b_k/c)$。而且当 $N\to\infty$ 时，由黎曼可积性的定义，我们有 $\sum_{k=0}^{cN-1}b_k\to N\int_0^1 f(x)\mathrm{d}x=N$。

5.3.1 小节

1.



2. 在习题 5.2.4-14 的记号下

$$L(n)-B(n)=\sum_{k=1}^{t}((e_k+k-1)2^{e_k}-(e_1+1)2^{e_k})+2^{e_1+1}-2^{e_t}=$$

$$2^{e_1}-2^{e_t}-\sum_{k=2}^{t}(e_1-e_k+2-k)2^{e_k}\geqslant$$

$$2^{e_1}-(2^{e_1-1}+\cdots+2^{e_1-t+1}+2^{e_t})\geqslant 0$$

当且仅当对于某个 $k>j\geqslant 0$ 有 $n=2^k-2^j$ 时，等式成立。[当合并是像在习题 5.2.4-23 中那样“由顶向下”完成时，比较的极大次数是 $B(n)$]。

3.当 $n>0$ 时，最小键码恰恰出现 k 次的结果数为 $\binom{n}{k}P_{n-k}$。于是，对于 $n>0$，$2P_n=\sum_k\binom{n}{k}P_{n-k}$，而且由等式 1.2.9-(10)，我们有 $2P(z)=e^zP(z)+1$。

另一个证明从这样一个事实得出：即 $P_n=\sum_{k\geqslant 0}\left\{{n \atop k}\right\}k!$，由于 $\left\{{n \atop k}\right\}$ 是把 n 个元素划分成 k 个非空部分的方式数，而且这些部分可以以 $k!$ 种方式排列。于是，由等式 1.2.9-(23)，$\sum_{n\geqslant 0}P_nz^n/n!=\sum_{k\geqslant 0}(e^z-1)^k=1/(2-e^z)$。

还有另一个证明，或许最有趣，它得自于：如果我们以一种稳定的方式，把这些元素排成序列，使得当且仅当 $K_i<K_j$ 或（$K_i=K_j$ 且 $i<j$）时 K_i 居于 K_j 之前，即可得之。在所有 P_n 个结果当中，如果排列 $a_n\cdots a_1$ 包含 k 个升高，则一个给定的安排 $K_{a_1}\cdots K_{a_n}$ 现在恰恰出现 2^k 次；因此 P_n 可以借助于欧拉数 $P_n=\sum_k\left\langle{n \atop k}\right\rangle 2^k$ 来表达。当 $z=2$ 时等式 5.1.3-(20)确立了所求的结果。

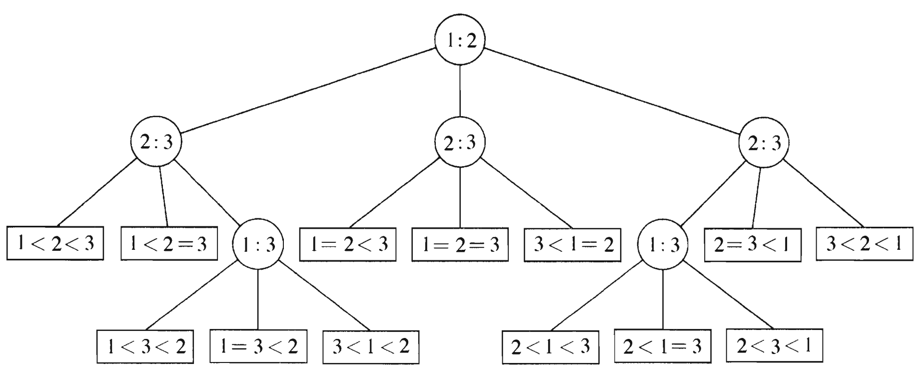
这个生成函数是由 A. Cayley[*Phil. Mag.* (4) **18** (1859)，374～378]在枚举一个不精确定义的树类时得到的。也见 P. A. MacMahon，*Proc. London Math. Soc.* **22** (1891)，341～344；J. Touchard，*Ann. Soc. Sci, Bruxelles* **53** (1933)，21～31。以及 O. A. Gross，*AMM* **69** (1962)，4～8；Gross 给出了有趣的公式 $P_n=\sum_{k\geqslant 1}k^n/2^{1+k}$，$n\geqslant 1$。

4.表示

$$2P(z)=\frac{1}{2}(1-\mathrm{i}\cot(\mathrm{i}(z-\ln 2)/2))=$$
$$\frac{1}{2}-\frac{1}{z-\ln 2}-\sum_{k\geqslant 1}\left(\frac{1}{z-\ln 2-2\pi \mathrm{i}k}+\frac{1}{z-\ln 2+2\pi \mathrm{i}k}\right)$$

产生一个收敛的级数 $P_n/n!=\frac{1}{2}(\ln 2)^{-n-1}+\sum_{k\geqslant 1}\Re((\ln 2+2\pi \mathrm{i}k)^{-n-1})$。

5.



6.$S'(n)\geqslant S(n)$，因为诸键码可以全都不同；于是我们必须证明 $S'(n)\leqslant$

$S(n)$。给定一个在不同的键码上花费 $S(n)$步的排序算法，我们可以通过定义 = 分支恒同于 < 分支来消除冗余性，从而构造一般情况的排序算法。当出现一个外部节点时，我们知道所有的相等关系，因为有 $K_{a_1} \leqslant K_{a_2} \leqslant \cdots \leqslant K_{a_n}$，而且对 $1 \leqslant i < n$ 已经明显地比较过 $K_{a_i}:K_{a_{i+1}}$。

M. Paterson 发现，如果键码的多重集是$(n_1, \cdots, n_m)$，则比较的次数可减少成为 $n \lg n - \sum n_j \lg n_j + O(n)$；见*SICOMP* **5** (1976)，2。通过如 Munro 和 Raman 在 *Lecture Notes in Comp. Sci.*，**519** (1991)，473～480 所建议的那样修改堆排序来处理相等的键码，则不用很多的辅助存储就几乎能达到这个下限了。

7. 见图 A-1。比较的平均次数是

$$(2+3+3+2+3+3+3+2 \cdot 3+3+3+3+2+3+3+2)/16 = 2\frac{3}{4}$$

8. 见图 A-2。比较的平均次数是 $3\frac{56}{81}$。

9. 如果所有的键码是相等的，则我们至少需要 $n-1$ 次比较才能发现所有键码相等这一事实。反之，$n-1$ 次就足够了，因为在把 K_1 同所有其它键码作了比较之后，我们总能导出最后的排序。

10. 设 $f(n)$为所求的函数，并设 $g(n)$是当 $k>0$ 和元素中恰有 k 个元素有已知值(0 或 1 时)，为对 $n+k$ 个元素进行排序所需平均比较次数。于是，$f(0) = f(1) = g(0) = 0$, $g(1) = 1$; $f(n) = 1 + \frac{1}{2}f(n-1) + \frac{1}{2}g(n-2)$, $g(n) = 1 + \min(g(n-1), \frac{1}{2}g(n-1) + \frac{1}{2}g(n-2)) = 1 + \frac{1}{2}g(n-1) + \frac{1}{2}g(n-2)$，其中 $n \geqslant 2$。(因此最好的策略是每当可能时，就比较两个未知的元素)。由此得出，对于 $n \geqslant 2$, $f(n) - g(n) = \frac{1}{2}(f(n-1) - g(n-1))$，以及对于 $n \geqslant 0$, $g(n) = \frac{2}{3}\left(n + \frac{1}{3}\left(1 - \left(-\frac{1}{2}\right)^n\right)\right)$。因此答案是对于 $n \geqslant 1$，

$$\frac{2}{3}n + \frac{2}{9} - \frac{2}{9}\left(-\frac{1}{2}\right)^n - \left(\frac{1}{2}\right)^{n-1}$$

(这个精确公式可以同信息论下限 $\log_3(2^n - 1) \approx 0.6309n$ 作比较。)

11. 二叉插入证明，对于 $n \geqslant m$, $S_m(n) \leqslant B(m) + (n-m)\lceil \lg(m+1) \rceil$。另一方面，$S_m(n) \geqslant \lceil \lg \sum_{k=1}^{m} \binom{n}{k} k! \rceil$，而且这渐近地等于 $n \lg m + O(((m-1)/m)^n)$；(参见等式 1.2.6～(53))。

12. (a)如果没有冗余的比较，则当实际上相等的键码第一次被比较时，我们可以任意地指定一个次序，因为不能从以前所做的比较中导出次序。(b)假定这株树对每个 0 和 1 的序列强排序；我们将证明，它对$\{1, 2, \cdots, n\}$的每个排列也强排序。假若不然，于是有一个排列，对于这个排列，它宣称$K_{a_1} \leqslant K_{a_2} \leqslant \cdots \leqslant K_{a_n}$，然而事实

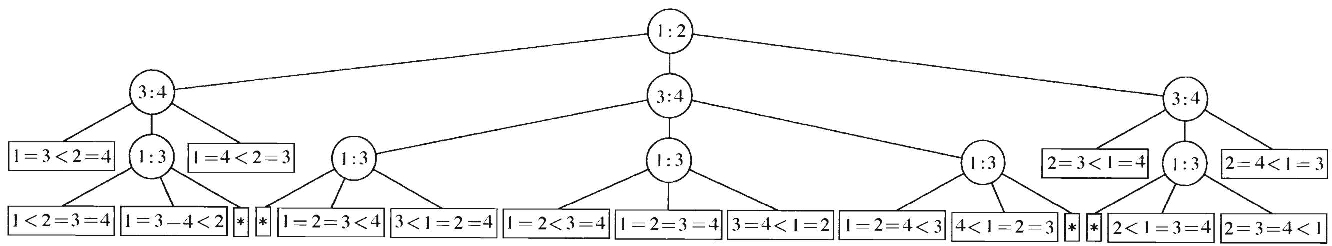


图 A-1　习题 7 的解

（“＊”表示一个不可能的情况）

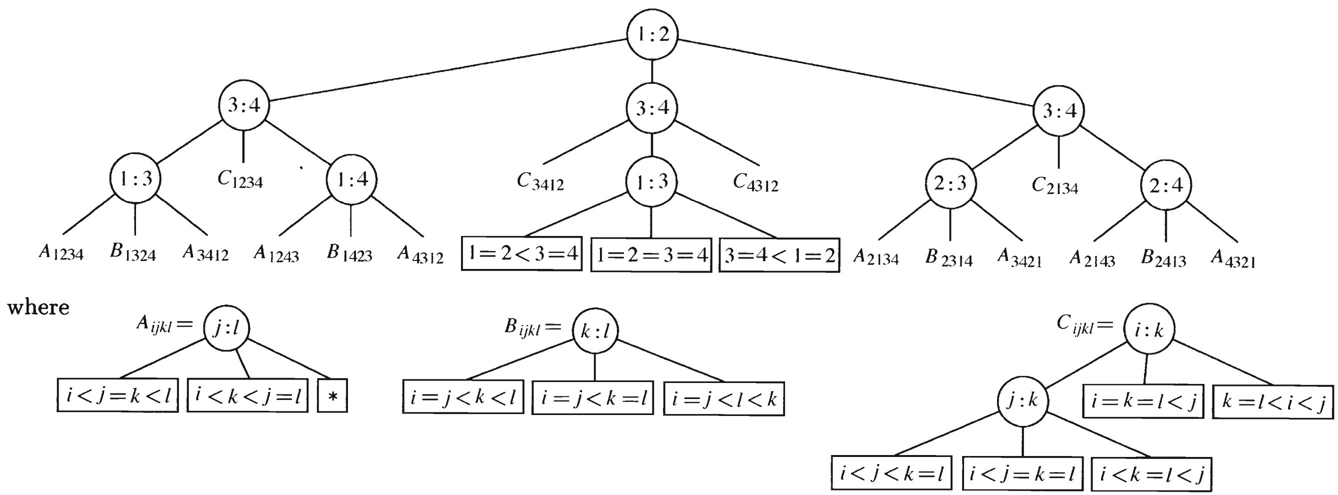


图 A-2　习题 8 的解

上对于某个 i，有 $K_{a_i} > K_{a_{i+1}}$。用 0 代替所有的 $< K_{a_i}$ 的元素，用 1 代替所有 $\geqslant K_{a_i}$ 的元素；由假设，当我们采取导致 $K_{a_1} \leqslant K_{a_2} \leqslant \cdots \leqslant K_{a_n}$ 的路径时，这个方法又应该能进行排序，矛盾。

13. 如果 n 为偶数，则 $F(n) - F(n-1) = 1 + F(\lfloor n/2 \rfloor) - F(\lfloor n/2 \rfloor - 1)$，所以我们必须证明 $w_{k-1} < \lfloor n/2 \rfloor \leqslant w_k$；由于 $w_{k-1} = \lfloor w_k/2 \rfloor$，这是明显的。如果 n 为奇数，则 $F(n) - F(n-1) = G(\lceil n/2 \rceil) - G(\lfloor n/2 \rfloor)$，所以我们必须证明 $t_{k-1} < \lceil n/2 \rceil \leqslant t_k$；由于 $t_{k-1} = \lceil w_k/2 \rceil$，这是显然的。

14. 由习题 1.2.4-42，和数为 $n\left\lceil \lg \frac{3}{4}n \right\rceil - (w_1 + \cdots + w_j)$，其中 $w_j < n \leqslant w_{j+1}$。后一个和数是 $w_{j+1} - \lfloor j/2 \rfloor - 1$。因此 $F(n)$ 可表达成 $n\left\lceil \lg \frac{3}{4}n \right\rceil - \lfloor 2^{\lfloor \lg(6n) \rfloor}/3 \rfloor + \left\lfloor \frac{1}{2}\lg(6n) \right\rfloor$（以及许多其它方式）。

15. 如果 $\left\lceil \lg \frac{3}{4}n \right\rceil = \lg\left(\frac{3}{4}n\right) + \theta$，则 $F(n) = n\lg n - (3 - \lg 3)n + n(\theta + 1 - 2^{\theta}) + O(\log n)$。如果 $\lceil \lg n \rceil = \lg n + \theta$，则 $B(n) = n\lg n - n + n(\theta + 1 - 2^{\theta}) + O(\log n)$。[注意 $\lg n! = n\lg n - n/(\ln 2) + O(\log n)$；$1/(\ln 2) \approx 1.443$；$3 - \lg 3 \approx 1.415$。]

17. $b_k < a_p < b_{k+1}$ 的情况数为

$$\binom{m-p+n-k}{(m-p)}\binom{p-1+k}{p-1}$$

而 $a_j < b_q < a_{j+1}$ 的情况数是

$$\binom{n-q+m-j}{n-q}\binom{q-1+j}{q-1}$$

18. 否，因为我们仅仅考虑在每个比较之下树的效率较低的分支。更有效的分支之一可以证明是更难于处理的。

20. 设 L 是出现有外节点的极大的级，而设 l 是极小的这样的级。如果 $L \geqslant l + 2$，则我们可以从级 L 撤销两个节点，并把它们放置在级 l 处的一个节点之下；这使外部路径长度减少 $l + 2L - (L - 1 + 2(l+1)) = L - l - 1 \geqslant 1$。反之，如果 $L \leqslant l + 1$，就设在级 l 上有 k 个外节点而在级 $l+1$ 上有 $N - k$ 个外节点，这里 $0 < k \leqslant N$。由习题 2.3.4.5-3，$k2^{-l} + (N-k)2^{-l-1} = 1$；因此，$N + k = 2^{l+1}$。不等式 $2^l \leqslant N < 2^{l+1}$ 现在表明 $l = \lfloor \lg N \rfloor$；这定义了 k 并给出外部路径长度(34)。

21. 设 $r(x)$ 是 x 的右子树的根。所有子树有极小的高度当且仅当对于所有的 x，$\lceil \lg t(l(x)) \rceil \leqslant \lceil \lg t(x) \rceil - 1$ 和 $\lceil \lg t(r(x)) \rceil \leqslant \lceil \lg t(x) \rceil - 1$。头一个条件等价于 $2t(l(x)) - t(x) \leqslant 2^{\lceil \lg t(x) \rceil} - t(x)$，而第二个条件等价于 $t(x) - 2t(l(x)) \leqslant 2^{\lceil \lg t(x) \rceil} - t(x)$。

22. 由习题 20，四个条件$\lfloor \lg t(l(x)) \rfloor, \lfloor \lg t(r(x)) \rfloor \geqslant \lfloor \lg t(x) \rfloor - 1$ 和 $\lceil \lg t(l(x)) \rceil, \lceil \lg t(r(x)) \rceil \leqslant \lceil \lg t(x) \rceil - 1$ 是必要和充分的。如同在习题 21 中那样论证，我们可以证明它们等价于所述条件。[Martin Sandelius, *AMM* **68** (1961), 133～144]。关于它的一个推广见习题 33。

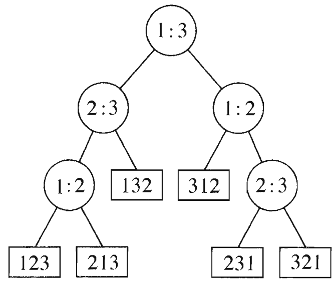
23. 多重表插入假设，诸键码被均匀地分布在一个已知的范围内，所以它不是满足本节中所考虑的限制的“纯粹比较”的方法。

24. 首先如同对五个元素排序那样进行，直到进行了五次比较，我们达到(6)中的配置之一之后为止。在头三种情况下，再用两次比较完成对五个元素的排序，然后插入第六个元素 f。在其它情况下，首先比较 $f:b$，把 f 插入到主链中，然后插入 c。[Picard, *Théorie des Questionnaires*, p. 116。]

25. 由于 $N = 7! = 5040$ 和 $q = 13$，在级 12 上将有 $8192 - 5040 = 3052$ 个外节点，且在级 13 个有 $5040 - 3152 = 1888$ 个外节点。

26. Ľ. Kollár[*Lecture Notes in Comp. Sci.*, **233**(1986), 449～457]已经介绍了一个杰出的方法来论证，这个最优的方法有 62416 的外部路径长度。

27.



是用两次比较识别两个最经常的排列的惟一方式，不过头一次比较产生 .27/.73 的分裂。

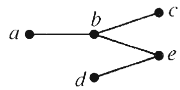
28. Lun Kwan 已经构造了一个其平均运行时间为 $38.925u$ 的 873 行的程序。它的极大运行时间为 $43u$；后者看来是最优的，因为它是 7 次比较，7 次测试，6 次装入，5 次存储的时间。

29. 我们必须至少作 $S(n)$次比较，因为不可能知道一个排列是偶的还是奇的，除非已经作了足够的比较来惟一地确定它。因为我们可以假定已经作了足够的比较，来把问题缩小到取决于 a_i 是否小于 a_j(对于某个 i 和 j)的两种可能性；这两种可能性之一是偶的，另一种是奇的。[另一方面，对于这个问题有一个 $O(n)$算法，这个算法简单地计算轮换的数目，而且全然不使用比较；见习题 5.2.2-2。]

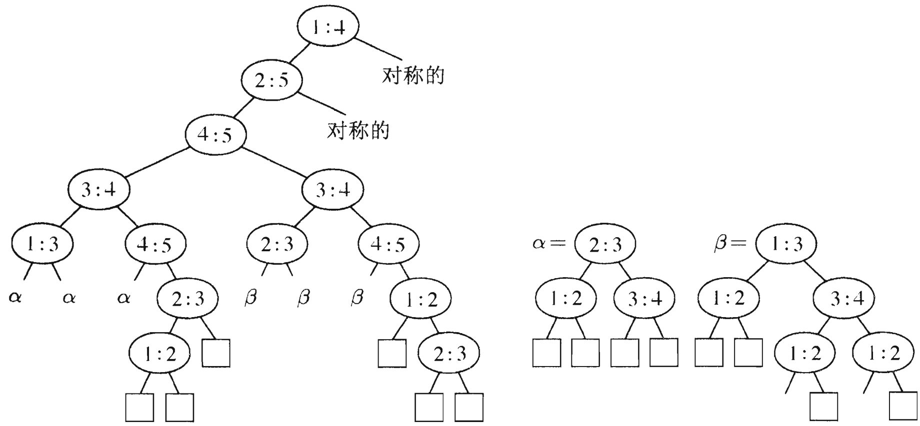
30. 由高度为 $S(n)$的最优比较树开始，从顶到底，重复地在标号为 $i:j$ 的节点的右子树中交换 $i \longleftrightarrow j$。把这个结果解释为一株比较-交换树，每个终端节点定义一个惟一的排列，这个排列可以通过至多 $n-1$ 次另外的比较-交换来进行排序(由习题 5.2.2-2)。

[比较-交换树的思想是由 T. N. Hibbard 给出的。]

31. 至少需要 8 个，因为高度为 7 的每一株树在 4 步之后，将在某个分支（或它



的对偶）产生配置，其中 $a \neq 1$。这个配置不能用 3 次另外的比较/交换操作来完成排序。另一方面，下列树达到了所希望的界（或许也是极小的平均比较/交换次数）：



33. 可应用于阶为 x 和分辨率为 1 的任何树的简单操作，也可被应用于产生另一株这样的树，即其加权路径长度不很大，其中对于某个 k，所有外节点位于级 k 和 $k-1$ 中，而且至多有一个外节点是非整数。而且，如果非整数节点存在，它位于级 k 上。任何这样的树的加权路径长度都有所述的值，因此这必定是极小的。反之，如果在任何实值查找树中(iv)和(v)成立，则通过归纳法可能证明，加权路径长度有所述的值，因为借助于根的两个子树的加权路径长度，存在有一株树的加权路径长度的简单公式。

36. [*Mat Zametki* **4** (1968), 511～518] 关于这个问题的进展的综述，以及关于我们总可以得到的下式

$$1 \leqslant T(G_1)/T(G_2) \leqslant \rho$$

的一个证明，其中常数 ρ 略小于 8/3，参见 S. Felsner 和 W. T. Trotter, *Combinatorics, Paul Erdös is Eighty* **1** (1993), 145～157。

5.3.2 小节

1. $S(m+n) \leqslant S(m)+S(n)+M(m,n)$。

2. 在对称次序下为第 k 个的内节点对应于比较 $A_1 : B_k$。

3. 策略 $B(1,l)$ 不比策略 $A(1,l+1)$ 更好，而策略 $B'(1,l)$ 不比 $A'(1,l-1)$ 更好；因此我们必须解递推式

$$.M.(1,n)=\min_{1\leqslant j\leqslant n}\max(\max_{1\leqslant l\leqslant j}(1+.M.(1,l-1)),\max_{j\leqslant l\leqslant n}(1+.M.(1,n-l)))$$
$$,n\geqslant 1;.M.(1,0)=0$$

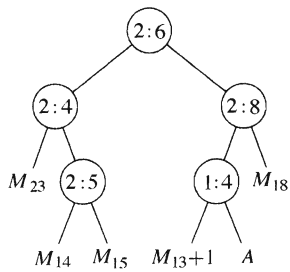
不难验证$\lceil\lg(n+1)\rceil$满足这个递推式。

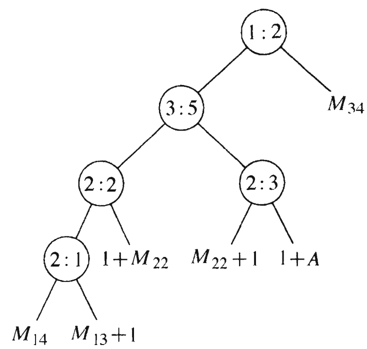
4. 否。[C. Christen, *FOCS* **19** (1978), 259～266]

6. 当 $j=i+1$ 时,除去 $i\leqslant 2$ 的情况外,可以使用策略 $A'(i,i+1)$。而当 $j\geqslant i+2$ 时,我们可以使用策略 $A(i,i+2)$。

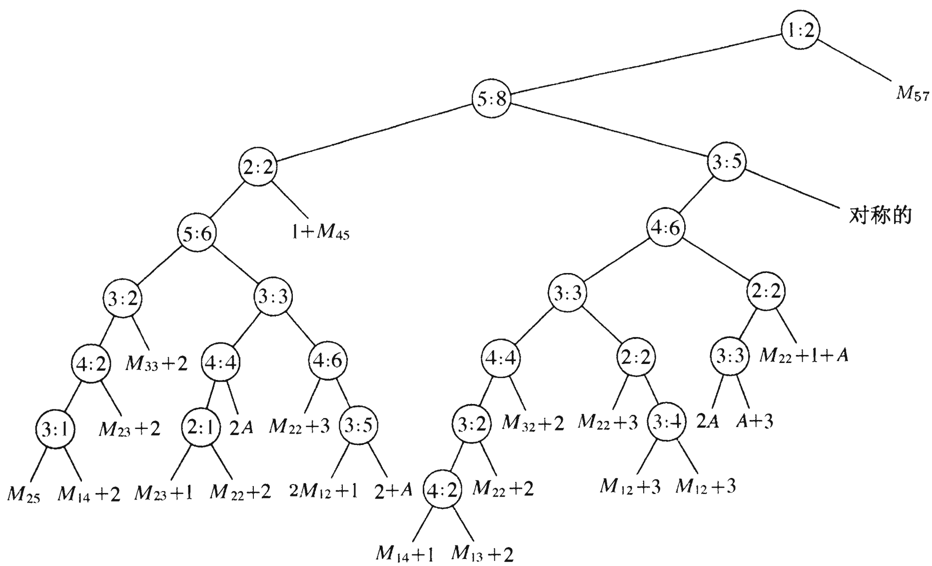
7. 为了在 n 个其它元素当中插入 $k+m$ 个元素,可独立地插入 k 个元素及 m 个元素。(当 k 和 m 很大时,本过程尚可改进,见习题 19)。

8,9. 在下列图式中,$i:j$ 表示比较 $A_i:B_j$,M_{ij}表示在$M(i,j)$步内把元素 i 同元素 j 合并,而 A 表示在三步内将型式  或  排序。





10.



11.如同在提示中那样,设 $n=g_t$。我们可以假定 $t\geqslant 6$。不失一般性,可设 $A_2:B_j$ 为头一次比较。如果 $j>g_{t-1}$,则结果 $A_2<B_j$ 将需要 $\geqslant t$ 个另加的步骤。如果 $j\leqslant g_{t-1}$,则结果 $A_2<B_j$ 将不成问题,所以仅仅需要研究 $A_2>B_j$ 的情况,而且当 $j=g_{t-1}$ 时我们得到最多的信息。如果 $t=2k+1$,则可能要把 A_2 同 $>B_{g_{t-1}}$ 的 $g_t-g_{t-1}=2^{k-1}$ 个元素合并在一起,而把 A_1 同 g_{t-1} 个其它元素合并在一起,但这要求 $k+(k+1)=t$ 个另外的步骤。另一方面,如果 $n=g_t-1$,则我们可以把 A_2 同 $2^{k-1}-1$ 个元素合并在一起,然后在 $(k-1)+(k+1)$ 个另外的步骤内,把 A_1 同 n 个元素合并在一起,因此 $M(2,g_t-1)\leqslant t$。

$t=2k$ 的情况要困难得多;注意 $g_t-g_{t-1}\geqslant 2^{k-2}$。在 $A_2>B_{g_{t-1}}$ 之后,假设我们比较 $A_1:B_j$。如果 $j>2^{k-1}$,则结果 $A_1<B_j$ 要求 $k+(k-1)$ 个另外的比较(太多)。如果 $j\leqslant 2^{k-1}$,则我们可以如同以前一样论证 $j=2^{k-1}$ 给出最多的信息。在 $A_1>B_{2^{k-1}}$ 之后,下一个同 A_1 的比较也可以同 $B_{2^{k-1}+2^{k-2}}$ 进行,然后同 $B_{2^{k-1}+2^{k-2}+2^{k-3}}$ 比;因为 $2^{k-1}+2^{k-2}+2^{k-3}>g_{t-1}$,我们就剩下把 $\{A_1,A_2\}$ 同 $n-(2^{k-1}+2^{k-2}+2^{k-3})$ 个元素合并在一起了。当然,我们不必马上进行同 A_1 的任何比较;而代之以比较 $A_2:B_{n+1-j}$。如果 $j\leqslant 2^{k-3}$,则考虑 $A_2<B_{n+1-j}$ 的情况,如果 $j>2^{k-3}$,则我们考虑 $A_2>B_{n+1-j}$。后一种情况要求至少再有 $(k-2)+(k+1)$ 个步骤。继续进行,我们发现仅有的可能有成果的路线是 $A_2>B_{g_{t-1}}$, $A_2<B_{n+1-2^{k-3}}$, $A_1>B_{2^{k-1}}$, $A_1>B_{2^{k-1}+2^{k-2}}$, $A_1>B_{2^{k-1}+2^{k-2}+2^{k-3}}$,但是在这种情况下我们恰恰剩下了 g_{t-5} 个元素!反之,如果 $n=g_t-1$,则这路线是行得通的。[*Acta Informatica* (1971),145~158。]

12.头一个比较必须是对于 $1\leqslant k\leqslant i$, $\alpha:X_k$ 或者(对称地)对于 $1\leqslant k\leqslant j$, $\beta:X_{n-k}$。在前一种情况下,如果 $\alpha<X_k$ 要我们继续做 $R_n(k-1,j)$ 次另外的比较;结果 $\alpha>X_k$ 要我们继续解决 $\alpha<\beta$, $Y_1<\cdots<Y_{n-k}$, $\alpha<Y_{i-k+1}$, $\beta>Y_{n-k-j}$ 的排序问题,其中 $Y_r=X_{r-k}$。

13.[*Computers in Number Theory* (New York: Academic Press, 1971), 263~269。]

14.[*SICOMP* **9** (1980),298~320。对于 $M(4,n)$ 的完全的解不久之后就由 J. Schulte Mönting 得到了。他还猜测对于 $M(5,n)$ 的解,在*Theor* . **14** (1981),19~37 上。]

15.将 m 加倍直到它超过 n 为止。这需要 $\lfloor\lg(n/m)\rfloor+1$ 次加倍。

16.当它是一个以上时,为除去(2,8),(3,6),(3,8)(3,10),(4,8),(4,10),(5,9),(5,10)之外的所有的 (m,n)。

17.假定 $m\leqslant n$ 且设 $t=\lg(n/m)-\theta$。则 $\lg\binom{m+n}{m}>\lg n^m-\lg m!\geqslant m\lg n-(m\lg m-m+1)=m(t+\theta)+m-1=H(m,n)+\theta m-\lfloor 2^\theta m\rfloor\geqslant H(m,n)+\theta_m-2^\theta m\geqslant H(m,n)-m$。(对于 $1\leqslant k<m$,不等式 $m!\leqslant m^m 2^{1-m}$ 由 $k(m-k)\leqslant$

$(m/2)^2$推出。)

19.首先合并$\{A_1,\cdots,A_m\}$和$\{B_2,B_4,\cdots,B_{2\lfloor n/2\rfloor}\}$。然后对于$1\leqslant i\leqslant\lceil n/2\rceil$,我们必须把诸奇元素$B_{2i-1}$插入到诸 A 的 a_i 当中,其中 $a_1+a_2+\cdots+a_{\lceil n/2\rceil}\leqslant m$。后一操作对于每个 i 至多需要 a_i 个操作,所以至多 m 次另外的比较就可完成这个工作。

20.应用(12)。

22.R. Michael Tanner[*SICOMP* **7** (1978),18～38]已经证明,一个"分离插入"算法平均至多作 $1.06\lg\binom{m+n}{m}$次比较。[Ľ. Kollár[*Computers and Artificial Int*. **5** (1986),335～344]已经研究了算法 H 的平均特性。

23.对手保持一个其元素 x_{ij}开始时全为 1 的 $n\times n$ 矩阵X。当这个算法问及是否 $A_i=B_j$ 时,对手置 x_{ij}成为 0。回答"否",除非 X 的永久式刚刚变成 0 了。在后一情况下,对手回答"是"(这是必须的,免得此算法立即结束!),并从 X 删去行 i 和列 j;得到的$(n-1)\times(n-1)$矩阵将有一个非 0 的永久式。对手继续照此办理,直到只剩下 0×0 矩阵。

如果要变成 0,我们可以重新安排诸行与列使得 $i=j=1$ 并且使此矩阵的对角线上全都为 1,但当 $x_{11}\leftarrow 1$ 时它的永久式仍变成 0;这时对于所有的 $k>1$,我们必定有 $x_{1k}x_{k1}=0$。由此得出,当对手第一次回答"是"时至少已删去 n 个 0,而当第二次回答"是"时已删去 $n-1$ 个 0,等等。仅仅在对非冗余的问题接受了 n 个"是"的回答之后,以及在提出了至少 $n+(n-1)+\cdots+1$ 个问题之后,这个算法才会结束[*JACM* **19** (1972),649～659]。类似的论证表明,当$|A|=m\leqslant n=|B|$时需要 $n+(n-1)+\cdots+(n-m+1)$个问题才能确定 $A\subseteq B$。

24.粗糙的预备性合并需要至多 $m+q-1$ 次比较,而且随继的插入每个至多需要 t 次。这些上限不可能减少。因此极大值和算法 H 的一样(参见(19))。

25.一般问题和其中每个 x_{ij}为 0 或 1 且 $x=\frac{1}{2}$的特殊情况一样困难。于是每个比较等价于考察二进位 x_{ij},而且我们要通过考察最少的二进位来确定整个矩阵。如果我们置 $x_{ij}=[A_i>B_{n+1-j}]$,则任何合并问题(1)对应于这样一个 0-1 矩阵。(N. Linial 和 M. Saks,在*J. Algorithms* **6** (1985),86～103 上把这个发现归功于 J. Shearer,一个类似的结果把查找同关于任何偏序的排序关连起来。)

5.3.3 小节

1.游戏者 11 失利于 05,所以知道 13 比 05,11,12 要差。

2.设 x 是第 t 个最大者,且 S 是使得所做比较不足以证明 $x<y$ 或 $y<x$ 的所有元素 y 的集合。存在同所作的所有比较相一致的一些排列,其中 S 的所有元素都小于 x;因为我们可以约定 S 的所有元素都小于 x 并把得到的偏序嵌入到一个线性序当中。类似地存在一些一致的排列,其中 S 的所有元素都大于 x。因此我们不知道 x 的秩,除非 S 是空的。

3.一个对手可以认为头一个比较的失利者是所有游戏者当中最差的。

4.假设最大的 $t-1$ 个元素是 $\{a_1,\cdots,a_{t-1}\}$。同这个假定相一致，为确定最大的 t 个元素的比较树中的任何通路，必定至少包括 $n-t$ 个比较以确定剩下的 $n-t+1$ 个元素的最大者。这样的通路至少有 $n-t$ 个二叉选择点，所以它们至少有 2^{n-t} 个。因此，对于最大的 $t-1$ 个元素的 n^{t-1} 种选择的每一个必定出现在这树的至少 2^{n-t} 个叶上。

5.事实上，由习题2，$W_t(n)\leqslant V_t(n)+S(t-1)$。

6.命 $g(l_1,l_2,\cdots,l_m)=m-2+\lceil\lg(2^{l_1}+2^{l_2}+\cdots+2^{l_m})\rceil$，并假定当 $l_1+l_2+\cdots+l_m+2m<N$ 时，$f=g$。我们将证明，当 $l_1+l_2+\cdots+l_m+2m=N$ 时，$f=g$。可以假定 $l_1\geqslant l_2\geqslant\cdots\geqslant l_m$。只有几种可能的方法来作头一次比较：

策略 A(j,k)，对于 $j<k$，比较组 j 的最大元素和组 k 的最大元素。这给出了关系

$$\begin{aligned}f(l_1,\cdots,l_m)\leqslant 1+g(l_1,\cdots,l_{j-1},l_j+1,l_{j+1},\cdots,l_{k-1},l_{k+1},\cdots,l_m)=\\ g(l_1,\cdots,l_{j-1},l_j,l_{j+1},\cdots,l_{k-1},l_j,l_{k+1},\cdots,l_m)\geqslant\\ g(l_1,\cdots,l_m)\end{aligned}$$

策略 B(j,k)，对于 $l_k>0$，比较组 j 的最大元素同组 k 的小元素之一。这给出关系

$$f(l_1,\cdots,l_m)\leqslant 1+\max(\alpha,\beta)=1+\beta$$

其中

$$\alpha=g(l_1,\cdots,l_{j-1},l_{j+1},\cdots,l_m)\leqslant g(l_1,\cdots,l_m)-1$$
$$\beta=g(l_1,\cdots,l_{k-1},l_k-1,l_{k+1},\cdots,l_m)\geqslant g(l_1,\cdots,l_m)-1$$

策略 C(j,k)，对于 $j\leqslant k$，$l_j>0$，$l_k>0$，比较取自组 j 的一个小元素同取自组 k 的一个小元素。对应的关系是

$$f(l_1,\cdots,l_m)\leqslant 1+g(l_1,\cdots,l_{k-1},l_k-1,l_{k+1},\cdots,l_m)\geqslant g(l_1,\cdots,l_m)$$

通过对于所有这些策略取右端极小值，即求出 $f(l_1,\cdots,l_m)$ 的值；因此 $f(l_1,\cdots,l_m)\geqslant g(l_1,\cdots,l_m)$。当 $m>1$ 时，策略 $A(m-1,m)$ 表明 $f(l_1,\cdots,l_m)\leqslant g(l_1,\cdots,l_m)$，因为当 $l_1\geqslant\cdots\geqslant l_m$ 时，$g(l_1,\cdots,l_{m-1},l_m)=g(l_1,\cdots,l_{m-1},l_{m-1})$。(证明：当 M 是 2^b 的一个正倍数时，对于 $0\leqslant a\leqslant b$ 有 $\lceil\lg(M+2^a)\rceil=\lceil\lg(M+2^b)\rceil$)。当 $m=1$ 时，使用策略 $C(1,1)$。

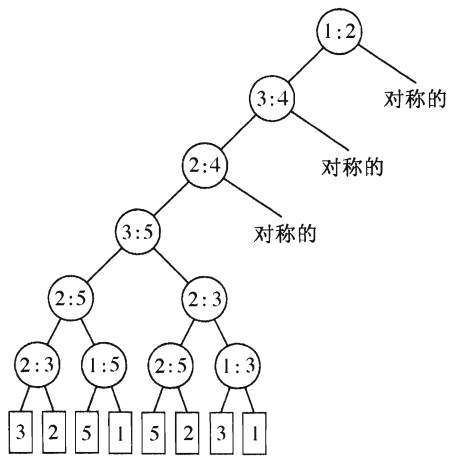
[S. S. Kislitsyn 的论文确定最优的策略 $A(m-1,m)$ 并以封闭的形式求出 $f(l,l,\cdots,l)$ 的值；f 的一般公式和这个简化了的证明是 Floyd 于1970年发现的。]

7.对于 $j>1$，如果 $j+1$ 在 α' 中，则 c_j 是1加上为选择 α' 的次一个最大元素所需要的比较数。如果 $j+1$ 在 α'' 中，则情况类似；而且 c_1 总为0，因为这树在端点处的样子总是相同的。

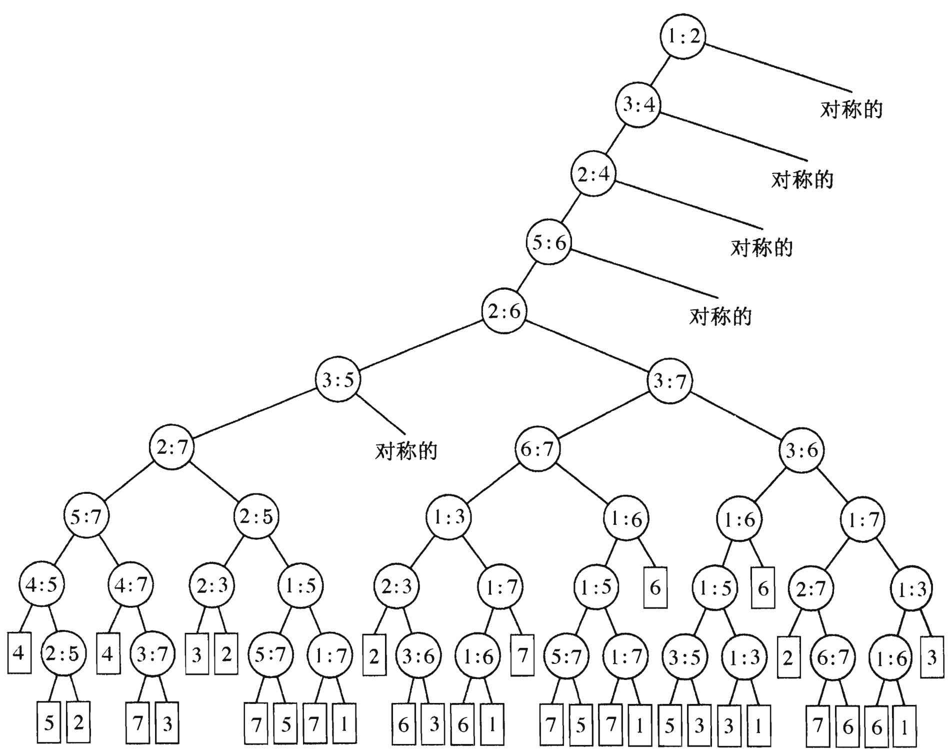
8.换言之，是否存在一株具有 n 个外节点的扩充的二叉树，使得从根到 $t-1$ 个最远的内节点的距离之和，小于完备的二叉树的相应的和？回答是否，因为不难证

明,对于所有的 α,$\mu(\alpha)$的第 k 个最大的元素至少是$\lfloor \lg(n-k) \rfloor$。

9.(所有路径都使用六次比较,但这个过程对于$\overline{V_3}(5)$不是最优的。)



10.(通过使用反复尝试法凭手工找出,并且使用习题 6 来帮助找出有成果的路线。)



11.参见*Information Processing Letters* **3** (1974),8～12。

12.在抛弃了$\{X_1,X_2,X_3,X_4\}$的最小者之后,我们有配置 •—• 加上 $n-3$ 个孤立的元素;这些当中的第三个最大者可以在 $V_3(n-1)-1$ 个另外的步骤之后求得。

13.在求得头 $f(n)$个元素的中间元素,比如说 X_j 之后,把它同其它每个元素进行比较;对于某个 k,这把诸元素分成近乎 $n/2-k$ 个小于X_j 的和$n/2+k$ 个大于 X_j 的两部分,剩下的是求较大的集合当中第$|k|$个最大或最小的元素,这需要 $n/2+O(|k|\log n)$次另外的比较。$|k|$的平均值(考虑诸点一致地分布于$[0\cdot\cdot 1]$中)是 $O(1/\sqrt{n})+O(n/\sqrt{f(n)})$。设 $T(n)$是当 $f(n)=n^{2/3}$时的平均比较数;则 $T(n)-n=T(n^{2/3})-n^{2/3}+n/2+O(n^{2/3})$,由此得出结果。

注意到这样一点是有趣的,即当 $n=5$ 时,这个方法平均仅需要 $5\frac{13}{15}$次比较,比习题 9 的树还稍好些。

14.一般地说,由于习题 2,通过求$\{X_1,\cdots,X_{n-1}\}$中的第 t 个最大者,并把它同 X_n 作比较,可以在 $U_t(n)\leqslant V_t(n-1)+1$ 次比较中求出第 t 个最大者。(对于更大的 t,Kirkpatrick 实际上证明(12)是 $U_t(n+1)-1$ 的下限。J. W. John,*SICOMP* **17** (1988),640～647 发现了对于 $U_t(n)$的一个改进了的下限。)

15.$\min(t,n+1-t)$。假定 $t\leqslant n+1-t$,如果当头 t 个字被读入时我们不把它们全部保存起来,则由于尚未知随后的值,我们可能忘记第 t 个最大者。反之,t 个单元已经足够了,因为我们可把新输入的元素同以前的第 t 个最大元素进行比较,当且仅当寄存器中的值是更大者时把它存起来。

16.这个算法由$(a,b,c,d)=(n,0,0,0)$开始并以$(0,1,1,n-2)$终止,如果对手避免"出人意外"的结果,则在每次比较之后,惟一可能的转换是从(a,b,c,d)到本身或者到

$(a-2,b+1,c+1,d)$,	如果 $a\geqslant 2$;
$(a-1,b,c+1,d)$ 或 $(a-1,b+1,c,d)$,	如果 $a\geqslant 1$;
$(a,b-1,c,d+1)$,	如果 $b\geqslant 2$;
$(a,b,c-1,d+1)$,	如果 $c\geqslant 2$。

由此得出,为了从(a,b,c,d)得到$(0,1,1,a+b+c+d-2)$,需要$\left\lceil\frac{3}{2}a\right\rceil+b+c-2$ 次比较。[参考文献*CACM* **15** (1972),462～464。在*FOCS* **16** (1975),71～74 中,Pohl 证明:这个算法也使平均的比较次数取极小值]。

17.首先对于最大者使用(6),然后对于最小者使用(6),注意其中有$\lfloor n/2\rfloor$次比较对两者是公共的。

18.对于所有充分大的 n,$V_t(n)\leqslant 18n-151$。

21.**步骤** 0.构造大小为 2^k 和 2^{k-t+1}的两棵淘汰树。

步骤 j. 对于 $1\leqslant j\leqslant t$。(这时我们输出了最大的 $j-1$ 个元素。剩下的元素,连同每个等于 $-\infty$ 的一组虚拟的占位者一起,现在出现在两棵淘汰树 A 和 B 中,其中 A 有 2^k个叶,而 B 有 2^{k-t+j}个。)设 a 是 A 的冠军,并且假定 a 已经击败 $a_0,a_1,\cdots,a_{k-1}$,其中 a_l 是 2^l 个元素的冠军。类似地,设 b 和 $b_0,b_1,\cdots,b_{k-t+j-1}$是 B 的冠军和次冠军。如果 $j=t$,则输出 $\max(a,b)$并停止。否则,通过引进 2^{k-t+j}个虚拟占位者在 B 的底部,而"增长"另一个级。这些虚拟占位者每一个在他们的首一轮中都已输给 B 的选手。(我们的策略将是:如果可能,通过把 B 和包含 $a_0,a_1,\cdots,a_{k-t+j}$ 的 A 的子树 A' 交换,而把 B 合并到 A;注意 A',像新近被扩大的 B 一样,是有 $2^{k-t+j+1}$个叶的一个淘汰树)。比较 b 和 $a_{k-t+j+1}$,然后比较其胜者和 $a_{k-t+j+2}$,等等,直到找到 $c=\max(b,a_{k-t+j-1},\cdots,a_{k-1})$为止。情况 1,$b<c$:输出 a 并把 B 同 A'交换。情况 2,$b=c$ 和 $b<a$:输出 a 并交换 B 和 A'。情况 3,$b=c$ 和 $b>a$:输出 b。在处理了这三种情况之后留给我们的是(可能是新的)淘汰树 A 和 B,其中 B 的冠军刚刚被输出。从 B 中删去该元素并以 $-\infty$ 代之,作任何必要的比较来恢复锦标淘汰结构(如同在树选择中那样)。这就完成了步骤 j。

步骤 0 做 $2^k-1+2^{k+1-t}-1$ 个比较,而步骤 t 作 1 次。步骤 $1,2,\cdots,t-1$ 各作至多 $k-1$ 次比较。但情况 2 除外,那时可能是 k 次。但每当情况 2 出现时,当下次是情况 1 或情况 2 时我们保留一次比较,因为 a_0 过后将是 $-\infty$。因此步骤 1 到 $t-1$ 总共至多作$(t-1)(k-1)+1$ 次比较。

由习题 3,当 $k\geqslant t\geqslant 2$ 时,对于所有 $n\leqslant 2^k+2^{k+1-t}$,我们有 $W_t(n)\leqslant n+(t-1)(k-1)$。如果 $n\geqslant 2^k+t-2$,习题 4 指出 $W_t(n)\geqslant n-t+\lceil\lg(2^k+t-2)^{\underline{t-1}}$,如果 $t\geqslant 3$, 它是 $n-t+(t-1)k+1$。因此当 $k\geqslant t\geqslant 3$ 时,对于 $2^k+t-2\leqslant n\leqslant 2^k+2^{k+1-t}$,这个方法是最优的。(如果 t 很大,对于 n 的若干较小的值也一样。)

当 $n\leqslant 2^k+2^{k+1-t}+t-2$ 和 $k\geqslant t\geqslant 3$ 时,当在步骤 $1,\cdots,t-2$ 的末尾重新构造 B 时,使用一个保留的元素而不使用 $-\infty$ 的一个类似的方法(见(11)的证明),可证明 $V_t(n)\leqslant n+(t-1)(k-1)$。[参见*J. Algorithms* **5** (1984),557~578。]

22. 一般地说,当 $2^r\cdot 2^k<n+2-t\leqslant(2^r+1)\cdot 2^k$ 和 $t<2^r\leqslant 2^t$ 时,以大小为 2^k 的 $t+1$ 个淘汰树开始的这一过程,将产生比(11)少$\lfloor(t-1)/2\rfloor$次比较,因为在(ii)中用来求极小的这许多比较在(iii)中可被"重用"。

23. 按照(15),当 $n\to\infty$ 时,量 $V_{\lceil n/2\rceil}(n)/n$ 以 2 为下限;但 D. Dor 和 U. Zwick 已经证明,实际的下限严格地大于 2,而上限小于上限 2.942[*SICOMP* **28** (1999),1722~1758;*SIAM J. Disc. Math.* **14**(2001),312~325]。他们还证明了一个渐近的上限

$$V_{\alpha n}(n)\leqslant\left(1+\alpha\lg\frac{1}{\alpha}+O\left(\alpha\log\log\frac{1}{\alpha}\right)\right)n$$

当 α 很小时,它离(15)不是太远[*Combinatorica* **16** (1996),41~58]。

24. 由等式(6),由于 $W_t(n)=n+O(t\log n)$,因此当 $t\leqslant\sqrt{n/\ln n}$时,提示中的命题肯定成立。假设对 n 该命题成立。并且设 u 和 v 在 $2n$ 个随机地有序的元素

的头 n 个中排位为 $t_- = \lfloor t - \sqrt{t \ln n} \rfloor$ 和 $t_+ = \lceil t + \sqrt{t \ln n} \rceil$。(最小的元素的排位为 1。)把其它 n 个元素和 v 作比较，并且还把那些小于 v 的那些元素和 u 作比较。在头 n 个元素中排位为 t 的一个元素 x 在整个中有排位 s 的概率 p_s 是 $\binom{s-1}{t-1}\binom{2n-s}{n-t}\Big/\binom{2n}{n}$。$s$ 的平均值是 $\sum sp_s = \frac{2n+1}{n+1}t$；这是 $< x$ 的元素的平均数，因此同 u 作比较的平均数是 $\binom{n}{n+1}t_+ = t + O(n\log n)^{1/2}$。设 u 和 v 在所有 $2n$ 个元素当中排位为 s_- 和 s_+，并设 $T_- = \lfloor 2t - \sqrt{2t \ln 2n}$ 和 $T_+ = \lceil 2t + \sqrt{2t \ln 2n} \rceil$。如果 $s_- < T_-$ 和 $s_+ > T_+$，通过在 u 和 v 之间由 $s_+ - s_- + 1$ 个元素中进行选择，我们能找出排位为 T_- 和 T_+ 的元素。我们将证明，有 $s_- > T_-$ 或 $s_- < T_- - 2\sqrt{n \ln n}$ 或 $s_+ < T_+$ 或 $s_+ > T_+ + 2\sqrt{n \ln n}$ 是非常不可能的；因此 $O(n\log n)^{1/2}$ 个另外的比较几乎就总是足够了。如果我们能够证明“非常不可能”指的是“对于所有充分大的 n，有概率 $O(n^{-1-\epsilon})$”，则通过对 n 用归纳法就可得出提示了。

注意 $p_{s+1}/p_s = s(n-s+t)/(s+1-t)(2n-s)$，当 s 从 t 增大成 $n+t$ 时减少，而且它是 $\leqslant 1$ 的当且仅当 $s \geqslant 2n(t-1)/(n-1)$；当 $s = \bar{s}(c) = 2t + ct(n-t)/n^{3/2}$ 时，它 $\leqslant 1 - \frac{1}{2}cn^{-1/2} + O(n^{-1})$。因此 $s \geqslant \bar{s}(c)$ 的概率是 $\leqslant 2c^{-1}n^{1/2}P_{\bar{s}(c)}(1 + O(n^{-1/2}))$。类似地，当 $s = \underline{s}(c) = 2t - 1 - c(t-1)(n+1-t)/n^{3/2}$ 时，$p_{s-1}/p_s < 1 - \frac{1}{2}cn^{-1/2} - O(n^{-1})$，所以 $s \leqslant \underline{s}(c)$ 的概率 $\leqslant 2c^{-1}n^{1/2}p_{\underline{s}(c)}(1 + O(n^{-1/2}))$。在我们需要的情况下，对于所有很大的 n，c 的相关的值 $\geqslant .55n^{3/2}(\ln n)^{1/2}t^{-1/2}(n-t)^{-1}$，而且 Stirling 近似蕴涵着 $p_{\bar{s}(c)}$ 和 $p_{\underline{s}(c)}$ 两者都是

$$O(n^{1/2}s^{-1/2}(2n-s)^{-1/2})\exp(-2sc^2(n-t)^2/n^3 - 2(2n-s)c^2t^2/n^3) \leqslant O(t^{-1/2}\exp(-4t(n-t)c^2/n^2)) \leqslant O(t^{-1/2}n^{-1.2})$$

因此 $O(n^{-1.2}(\log n)^{1/2})$ 的概率确实是非常不可能的。[一个类似的构造出现于 *CACM* **18** (1975)，165～172 中，但分析不正确。]

25. 给定一个选择算法和 $\{1, \cdots, n\}$ 的一个排列 π，设如果 $|\pi_i - t| > |\pi_j - t|$，则对于每个比较 $\pi_i : \pi_j$ 计 π_i 的费；如果 $|\pi_i - t| = |\pi_j - t|$，则两者各计 1/2。如果 $\pi_i < \pi_j \leqslant t$ 或 $\pi_i > \pi_j \geqslant t$，对 π_i 的计费称为有用的；否则它是无用的。设 x_k 是对 k 的所有计费，则比较总数是 $x_1 + \cdots + x_n$。显然 $x_t = 0$；但是对于所有 $k \neq t$，$x_k \geqslant 1$，因为不同于 t 的每个元素都有一个有用的计费。我们将证明对于 $0 < k < t$，$\mathrm{E}x_{t+k} + \mathrm{E}x_{t-k} \geqslant 3$。

命 $A_k(\pi) = [$对 $t+k$ 的头一次计费是无用的$]$。于是 $A_k(\pi) = 1 - A_{-k}(\pi')$，其中 π' 和 π 类似但以 $(t-k+1, \cdots, t+k, t-k)$ 分别代替元素 $(t-k, \cdots, t+k-1, t+k)$。因此 $\mathrm{E}A_k + \mathrm{E}A_{-k} = 1$。

命 $B_k(\pi)=[$对 $t+k$ 和 $t-k$ 的头一次计费都是$\frac{1}{2}$,而且 $t+k$ 接受它的第二次计费在 $t-k$ 之前$]$。还命 $C_k(\pi)=[x_{t+k}\geqslant 2+A_k]$。于是 $B_k(\pi)\leqslant C_k(\pi')$,其中 π 和 π' 类似,但是以$(t+k-1,t-k,\cdots,t+k-2)$代替元素$(t-k,t-k+1,\cdots,t+k-1)$。类似地,$B_{-k}(\pi)\leqslant C_{-k}(\pi'')$,其中 π'' 是从 π 通过把$(t-k+1,\cdots,t+k-1,t+k)$变成$(t-k+2,\cdots,t+k,t-k+1)$得到。由此得出,$\mathrm{E}B_k\leqslant \mathrm{E}C_k$ 且 $\mathrm{E}B_{-k}\leqslant \mathrm{E}C_{-k}$。

通过注意 $x_{t-k}+x_{t+k}\geqslant 2+A_k+A_{-k}-B_k-B_{-k}+C_k+C_{-k}$ 即可完成证明。[关于进一步的结果,参见*JACM* **36** (1989),270～279]。

(17)中的上限也有一个相匹配的下限,姚期智和姚储枫在*SICOMP* **11** (1982),428～447 中证明对于 $t>1$ 和 $n\geqslant(8t)^{18t}$, $\overline{V}_t(n)\geqslant n+\frac{1}{2}t(\ln\ln n-\ln t-9)$。

26.(a) 设两种类型分量的顶点被标记为 a;$b<c$。对手对非冗余的比较采取如下的动作:情况 1,$a:a'$,作出一个任意的决断。情况 2,$x:b$,比如说 $x>b$;此后对这个特定的 b 所作的所有未来的比较 $y:b$,都将得到 $y>b$,否则这些比较便通过对于 $U_t(n-1)$的一个对手来加以判定,其总数将为$\geqslant 2+U_t(n-1)$次比较。这个归结可被缩写成"令 $b=\min;2+U_t(n-1)$"。情况 3,$x:c$,令 $c=\max;2+U_t(n-1)$。

(b) 设新类型的顶点被标记为 $d_1,d_2<e$;$f<g<h>i$。情况 1,$a:a'$或 $c:c'$,任意决断。情况 2,$a:c$,比如说 $a<c$。情况 3,$x:b$,令 $b=\min;\ 2+U_t(n-1)$。情况 4,$x:d$,令 $d=\min;\ 2+U_t(n-1)$。情况 5,$x:e$,令 $e=\max;\ 3+U_{t-1}(n-1)$。情况 6,$x:f$,令 $f=\min;\ 2+U_t(n-1)$。情况 7,$x:g$,令 f 和 $g=\min;\ 3+U_t(n-2)$。情况 8,$x:h$,令 $h=\max;\ 3+U_{t-1}(n-1)$。情况 9,$x:i$,令 $i=\min;\ 2+U_t(n-1)$。

(c) 对于 $t=1$,我们有 $U_t(n)=n-1$,所以不等式成立。对于 $1<t\leqslant n/2-1$,使用归纳法和(b)。对于 $t=(n-1)/2$,使用归纳法和(a)。对于 $t=n/2$,$U_t(n-1)=U_{t-1}(n-1)$;使用归纳法和(a)。

27.(a)高度 h 满足 $2^h\geqslant\sum_l 1\geqslant\sum_l \Pr(l)/p=1/p$。

(b)如果 $r\leqslant t$,则在至少 $n-|S_0|-|T_0|=n-|S_0|-r$ 个触发之后,我们达到 A3。第 t 个最大元素将是 Q 的最小或者最大的元素,而且 Q 的元素彼此还未比较过,所以我们将需要至少另外的 $|Q|-1$ 个触发。如果 $|S_0|<q$,我们有 $|Q|=r$,否则我们有 $|Q|\geqslant|S_0|-|C(y_0)|+1\geqslant|S_0|-(q-r)+1$;所以在两种情况下都将作至少 $n-q$ 个触发。有 $n+1-t$ 个包含 $t-1$ 个最大元素的集合 T,这些元素是由一个给定的叶确定的,而且对于每一个这样的 T,达到该叶的概率是 0 或者 $2^{-f}\Big/\binom{n}{t}$,其中 $f\geqslant n-q$ 是对应于T的触发的次数。[这个对手在Bent和John的文章当中是含蓄的,见*STOC* **17** (1985),213～216]。

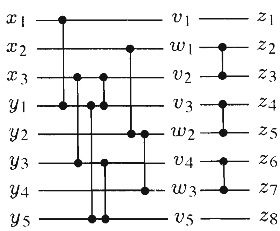
(c)如果 $t<r$，把 t 变成为 $n+1-t$；当 r 使右边取极大时，这将使 $t\geqslant r$，因为 r 将是 $O(\sqrt{n})$。如果对于所有 $y\in T_0$，有可能以 $|C(y)|>q-r$ 达到 A3，则除了它在 S 和 $T\setminus\{y_0\}$ 之间所作的至少 $(r-l)(q-r+1)$ 次比较之外，这个算法还将作 $n-1$ 个比较来把第 t 个最大元素同所有其它元素关联起来。

(d)选择 $r=\lceil\sqrt{m}\,\rceil$ 和 $q=2r-2$。(令 $q=r+\lfloor\sqrt{m}+\frac{1}{2}\rfloor-2$ 稍微好些；这个选择使(c)中导出的下限极大。)

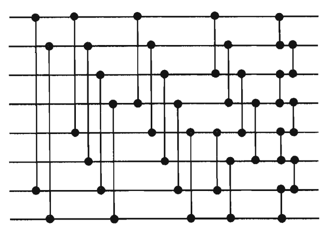
5.3.4 小节

1.(当 m 为奇数时，最好在 v_k 后边接上 $v_{k+1}, w_{k+1}, v_{k+2}, \cdots$，而不是像在图中那样接上 $w_{k+1}, v_{k+1}, w_{k+2}, \cdots$。由于被转换的行彼此相比较，因此这个改动是正确的。)

(3,5) 奇-偶合并



P_{ratt} 的八路排序

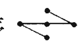


2.增量 h 需要 $2-[2h\geqslant n]$ 个级；对于 $n=8$ 见上面的图式。

3.对于 $m\geqslant 1$，$C(m,m-1)=C(m,m)-1$。

4.如果 $\hat{T}(6)=4$，则因为 $\hat{S}(6)=12$，每次将有三个比较动作。但是接着撤销底线和它的四个比较器将给出 $\hat{S}(5)\leqslant 8$，矛盾。[同样的论证产生 $\hat{T}(7)=\hat{T}(8)=6$]。Ian Parberry 通过穷尽的计算机查找已经证明 $\hat{T}(9)=\hat{T}(10)=7$；参见*Math. Systems Theory* **24** (1991)，101～116。]

5.如果 $n\geqslant 2$，设 $f(n)=f(\lceil n/2\rceil+1+\lceil\lg\lceil n/2\rceil\rceil)$；于是，对 n 用归纳法 $f(n)=(1+\lceil\lg n\rceil)\lceil\lg n\rceil/2$。

6.我们可以假定每个阶段作 $\lfloor n/2\rfloor$ 次比较(额外的比较也无妨)。因为 $\hat{T}(6)=5$，只需证明 $T(5)=5$。当 $n=5$ 时，在两个阶段之后，我们不能避免偏序  或 ，再用两个阶段不可能对它们的排序。

7.假定输入键码是 $\{1,2,\cdots,10\}$。关键码的事实在于，在进行了头 16 次比较之后，行 2,3,4 和 6 不可能包含 8 或 9，它们也不可能同时包含 6 和 7。(注意修改后的网络有延迟 8。)

8.定理 F 的直截了当的推广。

9.由习题 8，$\hat{M}(3,3)\geqslant\hat{S}(6)-2\hat{S}(3)$；$\hat{M}(4,4)\geqslant\hat{S}(8)-2\hat{S}(4)$；$\hat{M}(5,5)\geqslant$

$2\hat{M}(2,3)+3$；而且 $\hat{M}(2,3)\geqslant\hat{S}(5)-\hat{S}(2)—\hat{S}(3)$。类似地，$\hat{M}(3,4)=8$。但 $\hat{M}(3,5)$和 $\hat{M}(4,5)$等于多少呢？

10. 由定理 Z 中的证明方法可得提示。然后证明，偶子序列中 0 的个数减奇子序列中 0 的个数为 ±1 或 0。

11. (M. W. Green 给出的解。)在下列意义下，这个网络是对称的，即，每当 z_i 同 z_j 进行比较，则就有一个相应的比较 $z_{2^t-1-j}:z_{2^t-1-i}$。任何有能力对序列$\langle z_0,\cdots,z_{2^t-1}\rangle$排序的对称网络，也将对序列$\langle -z_{2^t-1},\cdots,-z_0\rangle$排序。

Batcher 已经发现，这个网络确实将对一个双调序列的任何循环移位$\langle z_j,z_{j+1},\cdots,z_{2^t-1},z_0,\cdots,z_{j-1}\rangle$排序。这是 0－1 原理的一个推论。

[当阶不是 2 的一个次幂时，对于双调排序器这些结果不成立。例如，图 52 不能对<0,0,0,0,0,1,0>进行排序。Batcher 对双调序列原来的定义更复杂，而且不如我们现在采用的定义有用]。

12. $x\vee y$ 是(考虑 0－1 序列)，但是 $x\wedge y$ 不是(考虑$\langle 3,1,4,5\rangle\wedge\langle 6,7,8,2\rangle$)。

13. 一个完全的洗牌有以 z_j 代替 z_i 的效果，这里 j 的二进表示就是由 i 的二进表示向右循环转动一位得到的(参见习题 3.4.2-13)。如果洗的是比较器而不是诸行；则这使得比较器的头一列作用于对偶 $z[i]$和 $z[i\oplus 2^{r-1}]$，下一列作用于 $z[i]$和 $z[i\oplus 2^{r-2}]$，…，第 t 列作用于$z[i]$和 $z[i\oplus 1]$，第 $t+1$ 列再次作用于 $z(i)$和 $z[i\oplus 2^{r-1}]$，等等。这里“$\oplus$”指二进表示的异或。这表明图 57 等价于图 56；在 s 个阶段之后，我们得到 2^s 个元素的一些组，它们的次序交替地呈正序和反序状。

C. G. Plaxton 和 T. Suel[*Math. Systems Theory* **27** (1994)，491～508]已经证明，任何这样的网络至少需要 $\Omega((\log n)^2/\log\log n)$级的延迟。

14. (a)对于 $i_s\neq k\neq j_s$，令 $y_{i_s}=x_{j_s}$，$y_{j_s}=x_{i_s}$，$y_k=x_k$；于是 $y\alpha^s=x\alpha$。(b)这是显然的除非集合$\{i_s,j_s,i_t,j_t\}$只有三个不同的元素；假设 $i_s=i_t$。于是，如果 $s<t$，则在$(\alpha^s)^t$ 和$(\alpha^t)^s$ 两者中，$s-1$ 个比较已经分别以(j_s,j_t,i_s)代替(i_s,j_s,j_t)。(c) $(\alpha^s)^s=\alpha$，和 $\alpha^1=\alpha$，所以我们可以假定 $s_1>s_2>\cdots>s_k>1$。(d)设 $\beta=\alpha[i:j]$；于是 $g_\beta(x_1,\cdots,x_n)=(\bar{x}_i\vee x_j)\wedge(g_\alpha(x_1,\cdots,x_i,\cdots,x_j,\cdots,x_n)\vee g_\alpha(x_1,\cdots,x_j,\cdots,x_i,\cdots,x_n))$。迭代这个恒等式就产生结果。(e)$f_\alpha(x)=1$ 当且仅当在 G_α 中没有从 i 到 j 的通路，其中 $x_i>x_j$。如果 α 是一个排序网络，则 α 的共扼也是；而且对于满足 $x_i>x_{i+1}$的所有 x，$f_\alpha(x)=0$。取 $x=e^{(i)}$；这表明，对于某个 $k_1\neq i$，G 有从 i 到 k_1 的一条有向弧。如果 $k_1\neq i+1$，$x=e^{(i)}\vee e^{(k_1)}$表明对于某个 $k_2\notin\{i,k_1\}$，$x=e^{(i)}\vee e^{(k_1)}$表明 G 有从 i 或 k_1 到 k_2 的一条有向弧。如果 $k_2\neq i+1$，如此继续，直到在 G 中找到从 i 到 $i+1$ 的一条通路为止。反之，如果 α 不是一个排序网络，令 x 是满足 $x_i>x_{i+1}$的一个向量，而且 $g_\alpha(x)=1$。某个共轭 α'有 $f_{\alpha'}(x)=1$，所以 $G_{\alpha'}$不可能有从 i 到 $i+1$ 的通路。[一般地说，对所有的 x，$(x\alpha)_i\leqslant(x\alpha)_j$ 当且仅当对于所有共轭于 α 的 α'，$G_{\alpha'}$有从 i 到 j 的一条有向通路。]

15. [1:4][3:2][1:3][2:4][2:3]。

16. 这过程显然终止。步骤 T2 的每一次执行有交换第 i_q 个和第 j_q 个输出的作用,所以这个算法的结果,是以某种方式排列输出行。由于得到的(标准)网络不改变输入 $\langle 1,2,\cdots,n\rangle$,输出行必然被恢复成它们原来的位置。

17. 通过习题 16 的算法使网络标准化;然后考虑输入序列 $\langle 1,2,\cdots,n\rangle$,我们看到标准选择网络必定把 t 个最大的元素放入到 t 个最高编号的行中;而且 $\hat{V}_t(n)$ 网络必然把第 t 个最大者放入行 $n+1-t$。应用 0－1 原理。

18. 定理 A 的证明表明 $\hat{V}_t(n)\geqslant(n-t)\lceil\lg(t+1)\rceil+\lceil\lg t\rceil$。

19. 网络 $[1:n][2:n]\cdots[1:3][2:3]$ 用 $2n-4$ 个比较器选择最小的两个元素;对于 $\hat{V}_2(n)$ 加 [1:2]。下限从定理 A 的证明得出(参见前面的答案)。

20. 首先注意,当 $n\geqslant 4$ 时,$\hat{V}_3(n)\geqslant\hat{V}_3(n-1)+2$;由对称性,可以假定头一个比较器是 $[1:n]$;在这之后,必然出现一个网络去选择 $\langle x_2,x_3,\cdots,x_n\rangle$ 中的第三个最大者,而另一个比较器接触行 1。另一方面,$\hat{V}_3(5)\leqslant 7$,因为四个比较器找到 $\{x_1,x_2,x_3,x_4\}$ 的 min 和 max,而剩下的是对三个元素排序。

21. 假的;例如,考虑两个网络[1:2][3:4][2:3][1:4][1:2][3:4]和[1:2][3:4][2:3][3:4][1:4][1:2][3:4]。(然而 N. G de Bruijn 在*Discrete Math.*, **9** (1974), 337 证明,新的比较器不会把在习题 36 的意义下是*原始*的排序网络弄乱。)

22. (a)对 α 的长度用归纳法,因为 $x_i\leqslant y_i$ 和 $x_j\leqslant y_j$ 蕴涵 $x_i\wedge x_j\leqslant y_i\wedge y_j$ 和 $x_i\vee x_j\leqslant y_i\vee y_j$。(b)对 α 的长度用归纳法,因为 $(x_i\wedge x_j)(y_i\wedge y_j)+(x_i\vee x_j)(y_i\vee y_j)\geqslant x_iy_i+x_jy_j$;[因此,$\nu(x\wedge y)\leqslant\nu(x\alpha\wedge y\alpha)$,这个发现是由 W. Shockley 给出的。]

23. 设 $x_k=1$ 当且仅当 $p_k\geqslant j$,$y_k=1$ 当且仅当 $p_k>j$;则 $(x\alpha)_k=1$ 当且仅当 $(p\alpha)_k\geqslant j$,等等。

24. 对于 l_i' 这个公式是显然的,而对于 l_j',如同在提示中那样取 $z=x\wedge y$,而由习题 21,发现 $(z\alpha)_i=(z\alpha)_j=0$。由习题 23 可知,加上额外的一些 1 到 z,即可看出存在一个排列 p,它使得 $(p\alpha')_j\leqslant\zeta(z)$。通过颠倒次序得出 u_i' 和 u_j' 的关系式。

25. (H. Shapiro 给出的解。)设 p 和 q 是排列且 $(p\alpha)_k=l_k$ 和 $(q\alpha)_k=u_k$。通过一系列的步骤,每步交换相邻的一对整数 $(i,i+1)$,有可能把 p 变换成 q;输入中这样一种交换对第 k 个输出的影响至多是 ± 1。

26. 存在有一一对应,它把 $P_n\alpha$ 的元素 $\langle p_1,\cdots,p_n\rangle$ 对应到"覆盖的序列" $x^{(0)}$ 覆盖 $x^{(1)}$ 覆盖…覆盖 $x^{(n)}$,其中 $x^{(i)}$ 在 $D_n\alpha$ 中;在这个对应中,当且仅当 $p_j=i$ 时 $x^{(i-1)}=x^{(i)}\vee e^{(j)}$。例如,$\langle 3,1,4,2\rangle$ 对应于序列 $\langle 1,1,1,1\rangle$ 覆盖 $\langle 1,0,1,1\rangle$ 覆盖 $\langle 1,0,1,0\rangle$ 覆盖 $\langle 0,0,1,0\rangle$ 覆盖 $\langle 0,0,0,0\rangle$。[姚期智发现,因此,只需对于 $\binom{n}{\lfloor n/2\rfloor}$ 个适当地选定的排列测试一个排序网络即可。例如排序 $\langle 4,1,2,3\rangle$,$\langle 3,1,4,2\rangle$,$\langle 3,4,1,$

2⟩⟨2,4,1,3⟩和⟨2,3,4,1⟩的任何 4 网络对任何事情进行排序。参见习题 6.5-1;也见习题 56]。

27. 这个原理成立因为$(x\alpha)_i$是x的第i个最小的元素。如果x和y表示其行是排好序的一个矩阵的不同的列,使得对所有的i,$x_i \leqslant y_i$,而且如果$x\infty$和$y\infty$表示对这些列排序的结果,则所述的原理表明,对于所有i,$(x\alpha)_i \leqslant (y\alpha)_i$,因为我们可以选择和$y$中的任何$i$个元素的处于相同的行上的$x$的$i$个元素。[我们曾使用这一原理,以证明谢尔排序的不变性性质,即定理 5.2.1K。关于这个思想的进一步的利用,见 David Gale 和 R. M. Karp 有趣的文章, *J. Computer and System Sciences* **6** (1972),103～115。关于列排序不弄乱排好序的行这一事实看来是在研究表格的处理时发现的;参见 Hermann Boerner, *Darstellung von Gruppen* (Springer,1955)第 5 章,§5。]

28. 如果$\{x_{i_1},\cdots,x_{i_t}\}$是$t$个最大的元素,则$x_{i_1}\wedge\cdots\wedge x_{i_t}$是第$t$个最大的。如果$\{x_{i_1},\cdots,x_{i_t}\}$不是$t$个最大的元素,则$x_{i_1}\wedge\cdots\wedge x_{i_t}$小于第$t$个最大的元素。

29. $\langle x_1\wedge y_1,(x_2\wedge y_1)\vee(x_1\wedge y_2),(x_3\wedge y_1)\vee(x_2\wedge y_2)\vee(x_1\wedge y_3),y_1\vee(x_3\wedge y_2)\vee(x_2\wedge y_3)\vee(x_1\wedge y_4),y_2\vee(x_3\wedge y_3)\vee(x_2\wedge y_4)\vee(x_1\wedge y_5),y_3\vee(x_3\wedge y_4)\vee(x_2\wedge y_5)\vee x_1,y_4\vee(x_3\wedge y_5)\vee x_2,y_5\vee x_3\rangle$。

30. 应用分配律和结合律把任何公式归约为诸$\wedge$的诸$\vee$的公式;然后用交换律,等幂律和吸收律导出规范形式。S_i正好是那样一些集合S,使得当$x_j=[j\in S]$时这个公式为 1,而对于S的任何真子集S',当$x_j=[j\in S']$时这公式为 0。

31. $\delta_4=166$。R. Church[*Duke Math. J.*, **6** (1940),732～734]求出$\delta_5=7579$,M. Ward[*Bull. Amer. Math. Soc.*, **52** (1946),423]求出$\delta_6=7828352$,以下的值是$\delta_7=2414682040996$,$\delta_8=56130437228687557907786$。[R. Chrtch, *Notices Amer. Math. Soc.* **12** (1965),724;J. Berman 和 P. Köhler, *Mitteilungen Math. Seminar GieBen*, **121** (1976),103～124;D. Wiedemann, *Order* **8** (1991),5～6]。显然对于δ_n没有简单的公式。D. Kleitman[*Proc. Amer. Math. Soc.* **21** (1969),677～682]使用一个极其复杂的论证证明了当$n\to\infty$时,$(\lg\delta_n)\Big/\binom{n}{\lfloor n/2\rfloor}\to 1$。

32. G_{t+1}也是所有串$\theta\psi$的集合,其中θ和ψ在G_t中,而且作为 0 和 1 的向量$\theta\leqslant\psi$。由此得出,G_t是所有 0 和 1 的串$z_0\cdots z_{2^t-1}$的集合,其中每当在 0-1 向量的意义下,i的二进表示"$\leqslant$"j的二进表示时即有性质$z_i\leqslant z_j$。G_t的每个元素$z_0,\cdots,z_{2^t-1}$,除了 00…0 和 11…1 之外,在$f(x_1,\cdots,x_t)=z[(x_1\cdots x_t)_2]$的对应之下,表示从$D_{2^t}$到$\{0,1\}$的一个$\wedge-\vee$函数$f(x_1,\cdots,x_t)$。

33. 如果这样一个网络存在,则对于某个函数f,我们将有$(x_1\wedge x_2)\vee(x_2\wedge x_3)\vee(x_3\wedge x_4)=f(x_1\wedge x_2,x_1\vee x_2,x_3,x_4)$或$f(x_1\wedge x_3,x_2,x_1\vee x_3,x_4)$或…或$f(x_1,x_2,x_3\wedge x_4,x_3\vee x_4)$。选择$\langle x_1,x_2,x_3,x_4\rangle=\langle x,\bar{x},1,0\rangle$,$\langle x,0,\bar{x},1\rangle$,$\langle x,$

$1,0,\overline{x}\rangle$,$\langle 1,x,\overline{x},0\rangle$,$\langle 1,x,0,\overline{x}\rangle$,$\langle 0,1,x,\overline{x}\rangle$表明不存在这样一个函数 f。

34.是;在证明了这点之后,你已经准备好着手处理图 49 中的 $n=16$ 的网络。(否则,你利用定理 Z 采用硬算的方法简单地测试所有 2^n 个二进位的向量。)

35.否则,其中仅有 i 和 $i+1$ 放错地方的排列将永远不被排序了。设 D_k 是在一个标准的排序网络中比较器$[i:i+k]$的个数。则 $D_1+2D_2+D_3 \geqslant 2(n-2)$,因为对于 $1\leqslant i\leqslant n-3$,从$\{i,i+1\}$到$\{i+2,i+3\}$以及$[1:2]$和$[n-1:n]$,必然有两个比较器。类似地 $D_1+2D_2+\cdots+kD_k+(k-1)D_{k+1}+\cdots+D_{2k-1}\geqslant k(n-k)$,这是一个由 J.M.Pollard 给出的公式。也可以证明:$2D_1+D_2\geqslant 3n-4$:如果我们对于所有的 j 删去形如$[j:j+1]$的头一些比较器,则对于 $1\leqslant i\leqslant n-2$ 必然至少还剩下一个比较器位于$\{i,i+1,i+2\}$之内。类似地 $kD_1+(k-1)D_2+\cdots+D_k\geqslant S(k+1)(n-k)+k(k-1)$。

36.(a)每个相邻的比较器都使反序数减少 0 个或 1 个,而且$\langle n,n-1,\cdots,1\rangle$有$\binom{n}{2}$个反序。(b)设 $\alpha=\beta[p:p+1]$,对 α 的长度用归纳法,进行论证如下:如果 $p=i$,则 $j>p+1$,而且$(x\beta)_p>(x\beta)_j$,$(x\beta)_{p+1}>(x\beta)_j$;因此$(y\beta)_p>(y\beta)_j$ 和$(y\beta)_{p+1}>(y\beta)_j$。如果 $p=i-1$,则或者$(x\beta)_p$ 或者$(x\beta)_{p+1}$要$>(x\beta)_j$;因此或者$(y\beta)_p$ 或者$(y\beta)_{p+1}>(y\beta)_j$。如果 $p=j-1$ 或 j,论证是类似的。对于其它的 p,论证是平凡的。

注意:如果 α 是一个原始排序网络,因此 α^R(在颠倒次序下的比较器)也是。关于(c)的推广和另一个证明,参见 N.G.de Bruijn,*Discrete Math.*, **9** (1974),333~339;*Indagationes Math.* **45** (1983),125~132。在后一篇论文中,de Bruijn 证明,一个原始排序网络对多重集合$\{n_1\cdot 1,\cdots,n_m\cdot m\}$的所有排列排序当且仅当它对单排列 $m^{c_m}\cdots 1^{c_1}$进行排序。对于排列 x 和 y 定义的关系 $x\leq y$ 指的是存在一个标准网络 α 使得 $x=y\alpha$,这叫做 Bruhat。限定于原始 α 的类似关系称为弱 Bruhat 次序(参见 5.2.1-44 题的答案)。

37.只需证明:如果每个比较器被一个交换操作所代替,则我们便得到一个"反序网络",它把$\langle x_1,\cdots,x_n\rangle$转换成为$\langle x_n,\cdots,x_1\rangle$。但在这种解释之下,不难来跟踪 x_k 的路线。(注意,排列 $\pi=(1\ \ 2)(3\ \ 4)\cdots(2n-1\ \ 2n)(2\ \ 3)(4\ \ 5)\cdots(2n-2\ \ 2n-1)=(1\ \ 3\ \ 5\ \ \cdots\ \ 2n-1\ \ 2n\ \ 2n-2\ \ \cdots\ \ 2)$满足 $\pi^n=(1\ \ 2n)(2\ \ 2n-1)\cdots(n-1\ \ n)$)。1954 年 H.Seward 简略地提到了奇偶转置排序;它曾经为 A.Grasselli[*IRE Trans*. **EC-11** (1962),483]和为 Kautz,Levitt 和 Waksman[*IEEE Trans*. **C-17**(1968),443~451]所讨论。关于这个网络的反射性实际上老早就由 H.E.Dudeney 在他的"青蛙难题"之一给出了[*Strand* **46** (1913),352,472; *Amusements in Mathematics* (1917),193]。

38.使用算法 5.1.4 I,把元素 $i_1,\cdots,i_N$ 插入到开始时为空的图表中,但有一个关键码性的改动:在步骤 I3 中仅当 $x_i\neq P_{i(j-1)}$时才置 $P_{ij}\leftarrow x_i$。当输入 $i_1\cdots i_N$ 定义

一个原始排序网络时，可以证明，仅当 $x_i+1=P_{ij}$ 时，在该步骤中 x_i 才将等于 $P_{i(j-1)}$。(算法中带括弧的断言需要加以修改。)如同在定理 5.1.4A 中那样，在把 i_j 插入到 P 中之后，置 $Q_{st}\leftarrow j$。在 N 步之后，图表 P 将总是在行 r 中包含 $(r,r+1,\cdots,n-1)$，而 Q 将是这样一个图表，通过向后进行，可以由它重新构造出序列 $i_1\cdots i_N$ 来。

例如，当 $n=6$ 时，序列 $i_1\cdots i_N=4\ \ 1\ \ 3\ \ 2\ \ 4\ \ 3\ \ 5\ \ 4\ \ 3\ \ 1\ \ 2\ \ 3\ \ 5\ \ 1\ \ 4$ 对应于

$P=$

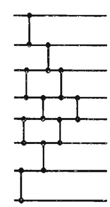
1	2	3	4	5
2	3	4	5	
3	4	5		
4	5			
5				

，$Q=$

1	4	5	8	13
2	6	7	15	
3	9	12		
10	11			
14				

。

参考文献：Q 的转置对应于补充网络 $[n-i_1:n-i_1+1]\cdots[n-i_N:n-i_N+1]$。A. Lascoux 和 M. P. Schützenberger, *Comptes Rendus Acad. Sci.* (Ⅰ) **295** (Paris, 1982), 629～633; R. P. Stanley, *Eur. J. Combinatorics* **5** (1984), 359～372; P. H. Edelman 和 C. Greene, *Advances in Math* **63** (1987), 42～99。原始排序网络的图式也对应于二维凸性的虚拟线和其它抽象的安排；关于进一步的信息，请见 D. E. Knuth, *Lecture Notes in Comp. Sci.* **606** (1992)。



39. 例如，当 $n=8$ 时，这样一个网络必须包括这里所示的比较器；所有其它的比较器对 10101010 都是低效的。然后像在习题 37 中那样，行 $\lceil n/3\rceil\cdots\lceil 2n/3\rceil=3\cdot\cdot6$ 对 4 元素进行排序。(本习题是以 David B. Wilson 的思想为基础的。)

注意：在对于一个给定的二进数串进行排序的极小长度原始排序网络和其形状由该二进数串定义的弯曲通路所限定的 Young 氏图表之间有一个一一对应。因此，习题 38 产生对 $(10)^{n/2}$ 进行排序的 $\binom{n/2+1}{2}$ 个比较器的原始网络和对 $n/2+1$ 个任意数进行排序的 $\binom{n/2+1}{2}$ 个比较器的原始网络之间的一一对应。如果一个原始网络对二进数串 $1^{n/2}0^{n/2}$ 进行排序，则我们可以作一个更强的断言：对于 $1\leqslant k\leqslant n/2$，在行 k 直到 $k+n/2$ (含 k 和 $k+n/2$) 上的子网络组成的所有它的"一半"，都是排序网络。(也可参见 de Bruijn 的定理，在习题 36 的答案中引用了它。)

40. 通过应用尾部不等式到 H. Rost 的一篇论文 (*Zeitschrift für Wahrscheinlichkeit Stheorie und Verwandte Gebiete* **58** (1981), 41～53) 中的命题 7 的有趣构造，并置 $b=\frac{1}{2}$，$a=\frac{1}{4}$ 和 $t=4n$，即可得出。

实验证明，达到任何原始排序网络的预期时间—不必是气泡排序—非常接近于

$2n^2$。奇怪的是，R. P. Stanley 和 S. V. Fomin 已经证明，如果以下面这样一种方式来非均匀地选择比较器$[i_k:i_k+1]$，即 $i_k=j$ 以$j\Big/\binom{n}{2}$的概率出现，则相应的预期时间就精确地成为$\binom{n}{2}H_{\binom{n}{2}}$。

42. 从某个输入到最大的输出，必定存在长度为$\lceil \lg n \rceil$或更长的一条通路（考虑定理 A 中的 m_n）；当把该输入置成∞时，在这条通路上的比较器有一个预先确定的特性，而且剩下的网络必定是一个$(n-1)$排序器。[*IEEE Trans on Computers*, **C-21** (1972), 612～613。]

45. 在 l 级之后，输入 x_1 可以至多在 2^l 个不同的位置上。在合并完成之后，x_1 可以在 $n+1$ 个不同的位置上。

46. [*J. Algorithms* **3** (1982), 79～88；以下的另一个证明是由 V. S. Grinberg 给出的。]我们可以假定 $1\leqslant m\leqslant n$ 而且每一阶段作 m 次比较，令 $l=\lceil (n-m)/2\rceil$，并假设我们正在把 $x_1\leqslant\cdots\leqslant x_m$ 同 $y_1\leqslant\cdots\leqslant y_n$ 合并。一个对手可以迫使$\lceil \lg(m+n)\rceil$阶段进行如下：在头一个阶段，某个 x_j 同一个元素 y_k 作比较，其中我们有 $k\leqslant l$ 或$k\geqslant l+m$。这个对手判定 $x_{j-1}<y_1$ 和 $x_{j+1}>y_n$；而且如果 $k\leqslant l$，则 $x_j>y_k$，而如果$k\geqslant l+m$，则 $x_j<y_k$。剩下的任务实质上是把 x_j 同$y_{k+1}\leqslant\cdots\leqslant y_n$ 或者同$y_1\leqslant\cdots\leqslant y_{k-1}$合并。所以至少剩下 $\min(n-k+1,k)\geqslant\min(n-l+1,l+m)=\lceil (m+n)/2\rceil$个结果。因此至少有$\lceil \lg\lceil (m+n)/2\rceil\rceil=\lceil \lg(m+n)\rceil-1$ 个随继的阶段是必需的。

48. 设 u 是$(x\alpha)_j$ 的最小元素，并设 $y^{(0)}$是 D_n 中任何一个这样的向量，它使得$(y^{(0)})_k=0$ 蕴涵着$(x\alpha)_k$ 包含一个$\leqslant u$ 的元素，$(y^{(0)})_k=1$ 蕴涵着$(x\alpha)_k$ 包含一个$>u$ 的元素。如果 $\alpha=\beta[p:q]$，则有可能找到一个满足同样条件，但是以 β 代替 α 的并且使得$y^{(1)}[p:q]=y^{(0)}$的向量 $y^{(1)}$。从$(y^{(0)})_i=1$，$(y^{(0)})_j=0$ 开始，我们最终有满足所需条件的向量 $y=y^{(r)}$。

G. Baudet 和 D. Stevenson 已经发现，把习题 37 和 48 结合在一起，产生出在 k 个处理器上只需$(n\ln n)/k+O(n)$个比较的简单的排序方法：首先对大小为$\leqslant\lceil n/k\rceil$的 k个子文件排序，然后使用 k 阶的“奇偶转置合并”在 k 次扫描中合并它们[*IEEE Trans*. **C-27** (1978), 84～87]。

49. $(x\vee\!\!\!\vee y)\vee\!\!\!\vee z$ 和 $x\vee\!\!\!\vee(y\vee\!\!\!\vee z)$两者都表示多重集合 $x+y+z$ 的最大的 m 个元素；$(x\wedge\!\!\!\wedge y)\wedge\!\!\!\wedge z$ 和 $x\wedge\!\!\!\wedge(y\wedge\!\!\!\wedge z)$表示最小的 m 个。如果 $x=y=z=\{0,1\}$，则$(x\wedge\!\!\!\wedge z)\vee\!\!\!\vee(y\wedge\!\!\!\wedge z)=(x\wedge\!\!\!\wedge y)\vee\!\!\!\vee(x\wedge\!\!\!\wedge z)\vee\!\!\!\vee(y\wedge\!\!\!\wedge z)=\{0,0\}$，而$\{0,0,0,1,1,1\}$的中间元素是$\{0,1\}$。三个元素的排序网络和习题 48 的结果意味着 $x+y+z$ 的中间元素可以表达成$((x\vee\!\!\!\vee y)\wedge\!\!\!\wedge z)\vee\!\!\!\vee(x\wedge\!\!\!\wedge y)$或$((x\wedge\!\!\!\wedge y)\vee\!\!\!\vee z)\wedge\!\!\!\wedge(x\vee\!\!\!\vee y)$或者在这些表达式中排列 x,y,z 后所得到的任何其它公式（对于中间元素似乎没有“对称”的公式）。

50. 等价地，由定理 Z，我们必定能找到满足在$[0\,..\,1]$中对有理值 x,y 的操作 $x\vee\!\!\!\vee y=\min(x+y,1)$，$x\wedge\!\!\!\wedge y=\max(0,x+y-1)$的所有恒等式。[这好比是由装满

x 的杯子中倒出尽可能多的液体到装满 y 的杯子中的操作，这是 J. M. Pollard 指出的]。所有这些恒等式可由四个公理的一个系统和 Lukasiewicz 的一个多值逻辑推导规则得到，见 Rose 和 Rosser，*Trans. Amer. Math. Soc.* **87** (1958)，1～53。

51. 设 $\alpha'=\alpha[i:j]$，并设 k 是一个 $\neq i,j$ 的指标。如果对所有 x，$(x\alpha)_i\leqslant(x\alpha)_k$，则 $(x\alpha')_i\leqslant(x\alpha')_k$；如果对所有 x，$(x\alpha)_k\leqslant(x\alpha)_i$ 且 $(x\alpha)_k\leqslant(x\alpha)_j$，则当 α 被代之以 α' 时同样的事实成立；如果对于所有的 x，$(x\alpha)_k\leqslant(x\alpha)_i$，则 $(x\alpha')_k\leqslant(x\alpha')_j$。这样我们看到，$\alpha'$ 至少有和 α 一样多的已知关系，如果 $[i:j]$ 不是冗余的，则还多加上一个[*Bell System Tech. J.* **49** (1970)，1627～1644。]

52. (a)考虑对诸 0 和诸 1 排序。设 $w=x_0+x_1+\cdots+x_N$。这个网络失效当且仅当在完备的 N-排序之前 $w\leqslant t$ 和 $x_0=1$。如果在这时 $x_0=1$，它必定开始时已是 1 了，而且对于 $1\leqslant j\leqslant n$，我们必定在开始时就有，对于 $0\leqslant k\leqslant m$，$x_{2j-1+2nk}=1$ 或者对于 $0\leqslant k\leqslant m$，$x_{2j+2nk}=1$；因此 $w\geqslant 1+(m+1)n=t$。所以失效意味着对于 $1\leqslant k\leqslant m$，$w=t$ 和 $x_j=x_{j+2nk}$ 以及对于 $1\leqslant j\leqslant n$，$x_{2j}=\bar{x}_{2j-1}$，而且对于 $1\leqslant j\leqslant m$，特殊的子网络必然把这样的输入转换成使得 $x_{2m+2n+j}=1$。

(b)例如，对于 $(y_1\vee y_2\vee\overline{y_3})\wedge(\overline{y_2}\vee y_3\vee\overline{y_4})\wedge\cdots$ 的特殊的子网络，利用 $x_{2j-1+2kn}$ 和 x_{2j+2kn} 来表示在第 k 个短句中的 y_j 和 $\bar{y}_j$，以及用 $x_{2m+2n+k}$ 来表示该短句本身，则可以是

$$[1+2n:2mn+2n+1][3+2n:2mn+2n+1][6+2n:2mn+2n+1]$$
$$[4+4n:2mn+2n+2][5+4n:2mn+2n+2][8+4n:2mn+2n+2]\cdots$$

53. 按照下列规则来画红线或蓝线

如果 i mod 4 是	则在情况(a)中的线 i	以及情况(b)中它是
0	红	红
1	蓝	红
2	蓝	蓝
3	红	蓝

现在观察由两个分开的网络组成的网络的头 $t-1$ 级，这两个网络一个是对于 2^{t-1} 条红线的，而另一个是对于 2^{t-1} 条蓝线的。如同在双调或奇-偶合并中那样，在第 t 级上的比较器完成一个合并网络。这就建立了对于 $k=1$ 的要求的结果。

红蓝分解也确立了 $k=2$ 的情况。因为如果输入是 4 有序的，则红线包含 2 有序的 2^{t-1} 个数，对于蓝线也一样，因此在 $t-1$ 级之后，我们得到

$$x_0y_0y_1x_1x_2y_2y_3x_3\cdots(\text{情况(a)})\text{ 或 }x_0x_1y_0y_1x_2x_3y_2y_3\cdots(\text{情况(b)})$$

最后结果

$$(x_0\wedge y_0)(x_0\vee y_0)(y_1\wedge x_1)(y_1\vee x_1)\text{ 或}$$
$$x_0(x_1\wedge y_0)(x_1\vee y_0)(y_1\wedge x_2)(y_1\vee x_2)\cdots$$

显然是 2 有序的。

现在对于 $k \geqslant 2$，我们可以假定 $k \leqslant t$。头 $t-k+2$ 级分解成 2^{k-2} 个大小为 2^{t-k+2} 分开的网络。由 $k=2$ 的情况知，它们每一个都是 2 有序的；因此在 $t-k+2$ 级之后这些线是 2^{k-1} 有序的。随后的级显然保持为 2^{k-1} 有序的，因为它们有阶 2^{k-2} 的“垂直的”周期。（我们可以想像在 $-1,-2,\cdots$ 线上的 $-\infty$ 和在 $2^t, 2^{t+1},\cdots$ 线上的 $+\infty$）。

参考文献：网络(a)首先是由 M. Dowd、Y. Perl、L. Rudolph 以及 M. Saks, *JACM* **36** (1989), 738～759 引进的；网络(b)是由 E. R. Canfield 和 S. G. Williamson, *Linear and Multilinear Algebra* **29** (1991), 43～51 引进的。指出这样一点是有趣的，在情况(a)中，我们有 $D_n\alpha = G_t$，其中 G_t 在习题 32 中定义[Dowd 等，定理 17]；因此，D_n 的映像本身不足于表征一个周期网络的特性。

54. 下列由 Ajtai、Komlós 和 Szemerédi[*FOCS* **33** (1992), 686～692]给出的构造表明如何使用四级 m^2 排序器来对 m^3 个元素进行排序：我们可以假设，被排序的元素是诸 0 和诸 1；对于 $0 \leqslant a, b, c < m$ 令线被编号为 $(a,b,c) = am^2 + bm + c$。头一级对于 $0 \leqslant k < m$，对线 $\{(a, b(b+k) \bmod m \mid 0 \leqslant a, b < m\}$ 进行排序；令 a_k 是 m^2 条线的第 k 组中 1 的个数。第二级对于 $0 \leqslant k < m$ 对线 $\{(a,b,k) \mid 0 \leqslant a, b < m\}$ 进行排序；于是在第 k 组中 1 的个数是

$$b_k = \sum_{j=0}^{m^2-1} \left\lfloor \frac{a_{(k-j) \bmod m} + j}{m^2} \right\rfloor$$

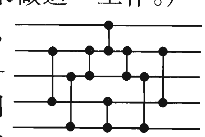
而且由此得出 $b_0 \leqslant b_1 + 1, b_1 \leqslant b_2 + 1, \cdots, b_{m-1} \leqslant b_0 + 1$。在第三级中，对于 $0 \leqslant k < m$，我们对线 $\{(k,a,b) \mid 0 \leqslant a, b < m\}$ 进行排序；在第 k 组中 1 的个数为

$$c_k = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} \left\lfloor \frac{b_i + km + j}{m^2} \right\rfloor$$

如果对于 $j < k$，$0 < c_{k+1} < m^2$，我们有 $c_k \leqslant \binom{m-1}{2}$ 且 $c_j = 0$。类似地，对于 $j > k+1$，如果 $0 < c_k < m^2$，我们有 $c_{k+1} \geqslant m^2 - \binom{m-1}{2}$ 和 $c_j = 0$。因此，对于 $0 < k < m$，对线 $m^2k - \binom{m-1}{2} \cdot\cdot m^2k + \binom{m-1}{2} - 1$ 进行排序的第四级将完成这个排序。

由此得出，四级 m 排序器将对 $f(m) = \lfloor \sqrt{m} \rfloor^3$ 个元素进行排序，而且 16 级将对 $f(f(m))$ 个元素进行排序。这就证明了所述结果。因为当 $m > 24$ 时，$f(f(m)) > m^2$。（这个构造并不“紧”，所以我们大概可以用比 16 级少得多的级来做这一工作。）

55. [如果 $P(n)$ 表示在一个排列网络中所需要的极小开关数，则显然 $P(n) \geqslant \lceil \lg n! \rceil$。通过稍微推广 L. J. Coldstein 和 S. W. Leibholz 给出-的一个构造，*IEEE Trans* . **EC-16** (1967), 637～641，人们能够证明 $P(n) \leqslant (P\lfloor n/2 \rfloor) + P(\lceil n/2 \rceil) + n - 1$，因此，对所有的



n，$P(n) \leqslant B(n)$，其中 $B(n)$ 是等式 5.3.1-(3)的二叉插入函数。M.W.Green 已经证明(未发表)$P(5)=8$]。

56.事实上，当 x 有 k 个 0 时，我们可以归纳地构造 α_x 使得 $x\alpha_x = 0^{k-1}101^{n-k-1}$。基础情况 α_{10} 是空。否则，下列四种情况中至少有一种适用，其中 y 未被排序：(1) $x = y0$，$\alpha_x = \alpha_y[n-1:n][n-2:n-1]\cdots[1:2]$。(2) $x = y1$，$\alpha_x = \alpha_y[1:n][2:n]\cdots[n-1:n]$。(3) $x = 0y$，$\alpha_x = \alpha_y^+[1:n][1:n-1]\cdots[1:2]$。(4) $x = 1y$，$\alpha_x = \alpha_y^+[1:2][2:3]\cdots[n-1:n]$。通过把每个比较器$[i:j]$变成$[i+1:j+1]$，从 α 得到网络 α^+。[参见郑文祯和 B. Ravikumar, *Discrete Math.* **81** (1990), 1～9]。这个构造使用了 $\binom{n}{2}-1$ 个比较器，可以使用更少些吗？

57.[见朱洪和 R. Sedgewick, *STOC* **14** (1982), 296～302。] 通过归纳法可以容易地验证所述的延迟时间。但当 $A(0,n)=A(m,0)=0$ 时，分析递推式

$$A(m,n) = A(\lfloor m/2\rfloor,\lceil n/2\rceil) + A(\lceil m/2\rceil,\lfloor n/2\rfloor) + \lceil m/2\rceil + \lceil n/2\rceil - 1$$

的问题更困难。

双调合并作 $B(m,n) = C'(m+n)$ 次比较；参见(15)。因此我们可以使用 $\{\lfloor m/2\rfloor + \lceil n/2\rceil, \lceil m/2\rceil + \lfloor n/2\rfloor\} = \{\lfloor (m+n)/2\rfloor, \lceil (m+n)/2\rceil\}$ 的事实来证明 $B(m,n) = B(\lfloor m/2\rfloor,\lceil n/2\rceil) + B(\lceil m/2\rceil,\lfloor n/2\rfloor) + \lfloor (m+n)/2\rfloor$。于是由归纳法 $A(m,n) \leqslant B(m,n)$。

令 $D(m,n) = C(m+1,n+1) + C(m,n) - C(m+1,n) - C(m,n+1)$。我们有 $D(0,n) = D(m,0) = 1$ 以及当 $m+n$ 为奇数时 $D(m,n)=1$。否则 $m+n$ 为偶数和 $mn \geqslant 1$，而且我们有 $D(m,n) = D(\lfloor m/2\rfloor,\lfloor n/2\rfloor) - 1$。因此，对于所有 m，$n \geqslant 0$，$D(m,n) \leqslant 1$。

对于 A 的递推式等价于对于 C 的递推式，除非当 m 和 n 都为奇数时。而且在该种情况下，通过归纳法，我们有 $A(m,n) \geqslant C(\lfloor m/2\rfloor,\lceil n/2\rceil) + C(\lceil m/2\rceil,\lfloor n/2\rfloor) + \lceil m/2\rceil + \lceil n/2\rceil - 1 = C(m,n) + 1 - D(\lfloor m/2\rfloor,\lfloor n/2\rfloor) \geqslant C(m,n)$。

命 $l = \lceil \lg \min(m+n)\rceil$。对于 $0 \leqslant k < l$，在偶奇递归的级 k 上，对于 $0 \leqslant j < 2^k$ 我们实施大小分别为 $(m_{jk}, n_{jk}) = (\lfloor (m+j)/2^k\rfloor, \lfloor (n+2^k-1-j)/2^k\rfloor)$ 的 2^k 个合并。递归的费用，$\sum_j(\lceil m_{jk}/2\rceil + \lceil n_{jk}/2\rceil - 1)$，是 $f_k(m) + f_k(n) - 2^k$；我们可以写 $f_k(n) = \max(n'_k, n-n'_k)$，其中 $n'_k = 2^k\lfloor n/2^{k+1} + 1/2\rfloor$ 是最接近于 $n/2$ 的 2^k 的倍数。由于 $0 \leqslant f_k(n) - n/2 \leqslant 2^{k-1}$，所以对于级 0 到 $l-1$，递归的总费用是介于 $\frac{1}{2}(m+n)l - 2^l$ 和 $\frac{1}{2}(m+n)l$ 之间。

最后，如果 $m \leqslant n$，对于 $0 \leqslant j < 2^l - m$，在级 l 上的 2^k 个合并 (m_{jl}, n_{jl}) 有 $m_{jl} = 0$，对于 j 的 m 个其它的值 $m_{jl} = 1$。由于 $A(1,n) = n$，因此级 l 的总费用是 $\sum_{k=n}^{m+n-1}\lfloor k/2^l\rfloor \leqslant \sum_{k=n}^{m+n-1} k/m = \frac{m-1}{2} + n$。

和双调合并不同，因此偶奇合并，最优比较次数 $\hat{M}(m,n)$ 是在 $O(m+n)$ 之内。我们的推导事实上表明，$A(m,n)=\sum_{k=0}^{l-1}(f_k(m)+f_k(n)-2^k)+g_l(m+n)-g_l(\max(m,n))$，其中 $g_l(n)$ 可以表达成 $\sum_{k=0}^{n-1}\lfloor k/2^l\rfloor=\lfloor n/2^l\rfloor(n-2^{l-1}(\lfloor n/2^l\rfloor+1))$ 的形式。

58. 如果 $h[k+1]=h[k]+1$，而且文件不是有序的，则在下次扫描时对于它必然发生某件事情；由习题 5.2.2-1，这减少了反序的数目，因此这个文件最终将成为排好序的。但是如果对于 $1\leqslant k<m$，$h[k+1]\geqslant h[k]+2$，则如果最小的键码开始时在 R_2 中，那么它将决不能移动到其适当的位置去。

59. 我们使用提示，并认为 $K_{N+1}=K_{N+2}=\cdots=1$。如果在步骤 j 中 $K_{h[1]+j}=\cdots=K_{h[m]+j}=1$，而且如果对于某个 $i>h[1]+j$，$K_i=0$，则我们必然有 $i<h[m]+j$，因为 1 的个数少于 n。假设 k 和 i 是使得 $h[k]+j<i<h[k+1]+j$ 以及 $K_i=0$ 的极小值。设 $s=h[k+1]+j-i$；我们有 $s<h[k+1]-h[k]\leqslant k$。在第 $j-s$ 步中，必定至少有 $k+1$ 个 0 已被读写头扫描过，因为 $K_i=K_{h[k+1]+j-s}$ 在该步被置成 0；s 步以后，在 $K_{h[1]+j}$ 和 K_i（包含）之间，至少剩下 $k+1-s\geqslant 2$ 个 0，同 i 的极小性矛盾。

第二次扫描放好其次 $n-1$ 个元素，等等。如果我们从排列 $N\ N-1\cdots 2\ 1$ 开始，则头一次扫描把它变成为 $N+1-n\quad N-n\cdots 1\quad N+2-n\cdots N-1\quad N$，因为每当 $1\leqslant h[1]+j$ 和 $h[m]+j\leqslant N$ 时，$K_{h[1]+j}>K_{h[m]+j}$；因此这个界是最好的。

60. 假设 $h[k+1]-s>h[k]$ 和 $h(k)\leqslant s$；如果最小的键码在 R_{n-s} 处开始，则它在位置 $R_i(i>1)$ 处结束。因此 $h[k+1]\leqslant 2h[k]$ 是必要的；由下述定理的特殊情况 $t=0$，它也是充分的。

定理 *如果 $n=N$，以及如果 $K_1\cdots K_N$ 是 $\{1,2,\cdots,n\}$ 的一个排列，且如果对于 $1\leqslant k<m$ 和 $0\leqslant i\leqslant t$，$h[k+1]\leqslant h[k]+h[k-i]+i$，则对于 $1\leqslant i\leqslant t+1$，一次排序扫描将置 $K_i=i$。*（约定，当 $k\leqslant 0$ 时设 $h[k]=k$）。

证明 对 t 用归纳法；如果到步骤 t 时键码 $t+1$ 不在读写头下边，则我们可以假定对于某个 $s>0$，它出现在位置 $R_{h[k+1]+t-s}$ 中，其中 $h[k+1]-s<h[k]$；因此 $h[k-t]+t-s>0$。但如果我们考察步骤 $t-s$，则这是不可能的，因为按假定这个步骤将把元素 $t+1$ 放置到位置 $R_{h[k+1]+t-s}$ 中，尽管至少有 $t+1$ 个较低的头正在工作。 ▌

（这个条件对于 $t=0,1$ 是必要的；但对于 $t=2$ 则不然。）

61. 如果数 $\{1,\cdots,23\}$ 正在被排序，则上道习题中的定理表明 $\{1,2,3,4\}$ 找到了它们真正的目的地。当诸 0 和诸 1 被排序时，可以验证，在步骤 -2，-1 和 0 中，不可能当所有的读写头都读 0 时，所有不处于这些头之下的位置都包含 1；因此上题的证明可被推广以表明 $\{5,6,7\}$ 找到了它们真正的目的地。最后，由习题 53 中的论证，$\{8,\cdots,23\}$ 必然被排序。

63. 当 $r \leqslant m-2$ 时，读写头把串 $0^p 1^1 0 1^3 0 1^7 0 \cdots 0 1^{2^r-1} 0 1^q$ 变成为 $0^{p+1} 1^1 0 1^3 0 1^7 0 \cdots 0 1^{2^{r-1}-1} 0 1^{2^r-1+q}$；因此需要 $m-2$ 次扫描。[当读写头在位置 $1,2,3,5,\cdots,1+2^{m-2}$ 处时，Pratt 发现了一个类似的结果：串 $0^p 1^a 0 1^{2^b-1} 0 1^{2^{b+1}-1} 0 \cdots 0 1^{2^r-1} 0 1^q$，$1 \leqslant a \leqslant 2^{b-1}$，变成 $0^{p+1} 1^{a-1} 0 1^{2^b-1} 0 1^{2^{b+1}-1} 0 \cdots 0 1^{2^{r-1}-1} 0 1^{2^r+q}$，因此对于这个头的序列，在最坏情况下至少需要 $m - \lceil \log_2 m \rceil - 1$ 次扫描。后一个头序列是特别有趣的，因为它已经被用作 P. N. Amstrong 发明的非常精巧的排序设备的基础。[参见 *U. S. Patent* **3399383** (1965)。Pratt 猜测，对于所有输入，这些输入序列提供了真正最坏的情况。]

64. 在快速排序期间，每个键码 $K_2, \cdots K_N$ 都同 K_1 作过比较；设 $A = \{i \mid K_i < K_1\}$，$B = \{j \mid K_j > K_1\}$。随后的操作独立地对 A 和 B 快速排序；在快速排序和有限制的一致算法中，对于 A 中的 i 和 B 中的 j，禁止所有 $K_i : K_j$ 的比较，而无限制的一致算法不禁止其它比较。

在这种情况下，我们甚至可以进一步限制这个算法，省略情况 1 和情况 2 使得仅当明显地做比较时，才把弧加到 G 中，而且当测试冗余性时仅考虑长度为 2 的路径。解决这个问题的另一个办法，是考虑 6.2.2 小节的等价树插入排序算法，它正好以相同的次序进行与一致算法相同的比较。

65. (a) K_{a_i} 同 K_{b_i} 进行比较的概率，是 c_i 个其它的特定键码不处于 K_{a_i} 和 K_{b_i} 之间的概率；这是随机地从 $\{1,2,\cdots,c_i+2\}$ 中选择的两个数是相连的概率，也就是

$$(c_i+1) \Big/ \binom{c_i+2}{2}$$

(b) c_i 的头 $n-1$ 个值为 0，然后是 $(n-2)$ 个 1，$(n-3)$ 个 2，等等；因此平均值是 $2\sum_{k=1}^{n}(n-k)/(k+1) = 2\sum_{k=1}^{n}((n+1)/(k+1)-1) = 2(n+1)(H_{n+1}-1)-2n$。

(c) 合并的"两部分构成的"本性表明，对于这个序列来说有限制的一致算法和一致算法是相同的。对包含顶点 N 的对偶来说，诸 c 分别等于 $0,1,\cdots,N-2$；所以平均比较次数同快速排序完全相同。

66. 否；当 $N=5$ 时，没有以(1,5)(1,2)(2,3)(3,4)(4,5)结束的对偶序列会要求作 10 次比较。[一个有趣的研究问题：对于所有的 N，试找出一个(有限制的)一致排序方法，使其最坏情况尽可能地好]。

67. (Gil Kalai 已经正式宣告使用同他在 *Graphs and Combinatorics* **1** (1985)，65～79 中的论文有关的方法，他已给出对于有限制情况的极小性的证明。然而，他的证明还未被发表。)

68. 每次扫描时一个项至多失去一个反序，所以极小的扫描次数至少是在输入排列中任何项的极大反序数。气泡排序策略达到这个界，因为每次扫描都使每个有反序的项的反序计数减 1(参见习题 5.2.2-1)。可能需要另外的扫描来确定排序是否完备，但是这个习题的行文允许我们忽略这样的考虑。

也许不幸的是,通过自动机研究计算复杂性得到的头一个结果,是确定了一个排序的方法的“最优性”,而从程序设计的观点看来,它竟如此之低劣!同随机数生成的历史相类似,当从一个特定的观点看来某些生成程序是“最优的”,因而被建议广泛采用时,实际上却是倒退了好多步。(参见等式 3.3.8-(39)下面的注释。)教益在于最优性的结果通常大量地依赖于抽象模型;尽管这些结果是非常有趣的,但在实用中必须明智地来应用它们。

[Demuth 曾考虑对于一个 r 寄存器机器(保存 r 的一个因子)和对于一个类图灵机的推广。在这种机器中扫描方向可随意地在左右和右左之间摆动。他发现,后一种类型的机器可以作直接插入以及鸡尾混合排序;但是任何这样的 1-寄存器机器平均必须通过至少$\frac{1}{4}(N^2-N)$个步骤,因为每步至多使总的反序数减少 1。最后他考虑了 r-寄存器随机存取机器以及极小-比较排序的问题。他的论文的这些部分,已在*IEEE Transactions* **C-34**(1985),296～310 中重新印刷。]

5.4 节

1.我们可以省略内部排序阶段,但是一般地这样将会慢得多,因为它将增加每块数据在外存上读和写的次数。

2.诸路段如同在(1)中那样分布,然后带 3 被置为 $R_1\cdots R_{2000000}$;$R_{2000001}\cdots R_{4000000}$;$R_{4000001}\cdots R_{5000000}$。在重绕所有的带之后,一个“一路合并”置 T_1 和 T_2 分别成为(2)中 T_3 和 T_4 的内容。然后 T_1 和 T_2 被合并到 T_3,而信息被复写回去并再次被合并,总共进行五次扫描。一般地说,这个过程和四条带的平衡合并类似,但是在每次合并扫描之间有拷贝扫描,所以共实施了两倍减 1 次扫描。

3.(a)$\lceil\log_P S\rceil$。(b)$\log_B S$,其中 $B=\sqrt{P(T-P)}$称为“有效的合并能力”。当 $T=2P$ 时,有效能力是 P;当 $T=2P-1$ 时,有效能力是$\sqrt{P(P-1)}=P-\frac{1}{2}-\frac{1}{8}P^{-1}+O(P^{-2})$,比$\frac{1}{2}T$ 稍小些。

4.$\frac{1}{2}T$。如果 T 是奇数而且 P 一定是一个整数,则$\lceil T/2\rceil$和$\lfloor T/2\rfloor$给出相同的极大值。按照习题 3,最好有 $P\geqslant T-P$,所以对于平衡合并我们将选择 $P=\lceil T/2\rceil$。

5.4.1 小节

$$1.\ 087\quad 154\quad 170\quad 426\begin{cases}503\begin{cases}503 & \infty\\ 908 & \infty\end{cases}\\ 426\begin{cases}426 & 653 \quad \infty\\ 612 & \infty\end{cases}\end{cases}$$

2. 路径 [061]—(512)—(087)—(154)—(061) 将被改变成为 [612]—(612)—(512)—(154)—(087)。(沿着这条路径我们实际上进行了从一个“气泡排序”!)

3. and fourscore our seven years/ago brought fathers forth on this/a conceived continent in liberty nation new the to/and dedicated men proposition that all are created equal.

4. (这个问题稍微含混些;按这种解释,在水库快溢出之前我们不清除内存。) and fourscore on our seven this years/ago brought continent fathers forth in liberty nation new to/a and conceived dedicated men proposition that the/all are created equal。

5. 假的。对于所有 $P \geqslant 1$,定义具有 P 个外节点的完备的二叉树。

6. 在步骤 R6 的开始处插入“如果 T = LOC($X[0]$),则转到 R2,否则”并从步骤 R7 删去类似的短句。

7. 没有输出,且 RMAX 一直等于 0。

8. 如果最初 P 个实际的键码的任何一个是 ∞,则这些记录将丢失。为了避免 ∞,我们可以做这个程序的两个几乎相同的副本;头一个副本省略包含在步骤 R4 中的 LASTKEY 的测试,而且当在步骤 R3 中头一次有 RQ≠0 时它跳到第二个副本处。第二个副本不需要步骤 R1,而且在步骤 R3 中它决不需要测试 RQ。

9. 例如,假定当前的路段是递增的,而下一个将是递减的。则算法 R 的诸步骤除开下列一处要改动外,将正确地工作:在步骤 R6 中,如果 RN(T) = RQ > RC,则颠倒对于 KEY(LOSER(T))和 KEY(Q)的测试。

当 RC 改变时,步骤 R4 和 R6 的键码测试应当适当地改变。

10. 令 $j \equiv$ LOC($X[j]$)。如果我们首先置 LOSER($\cdot 0$)←Q 以及 RN($\cdot 0$)←RQ,则算法 R 的机制确保了每当我们达到步骤 R3 时下列条件为真:LOSER($\cdot 0$), …, LOSER($\cdot(P-1)$)的值是 $\{\cdot 0, \cdot 1, \cdots, \cdot(P-1)\}$ 的一个排列;而且存在诸指针 {LOSER($\cdot j$) | RN($\cdot j$) = 0} 的一个排列,它对应于一个实际的锦标赛。换言之,当 RN($\cdot j$) = 0时,KEY(LOSER($\cdot j$))的值是无关紧要的;我们可以在他们当中对“失利者”进行排列。在 P 步之后,所有RN($\cdot j$)都将非 0,所以整个树将是一致的。(对于提示的回答是“是的”。)

追求纯正者可能抱怨,此算法比较未曾初始化的 KEY 值。如果这样的行为太使人震惊,可以通过在比如说步骤 R1 中置所有 KEY 值成为 0 而避免它。

11. 真。(两个键码都取自于定理 K 的证明中的相同的子序列。)

12. 当头一个路段已经结束时,保留在内存中的键码一般比平均值更小,因为它们不使它进入头一个路段当中。于是第二个路段可以输出更多的较小的键码。

14. 假定在扫雪车达到它的稳定状态后,它处于随机点 u, $0 \leqslant u < 1$,雪突然停止,则倒数第二个路段包含$(1+2u-u^2)P$ 个记录,而最后的路段包含 $u^2 P$ 个记录。对它乘以 du 进行积分,就得出倒数第二个路段是$\left(2-\frac{1}{3}\right)P$ 个记录的平均时间,最后路段是$\frac{1}{3}P$ 个记录的平均时间。

15. 假的;最后的路段可以是任意长的,但是仅当输入被穷尽,而所有内存中的记录都属于同一个路段这种相当稀少的情况下才有可能。

16. 当且仅当每个元素都有少于 P 个反序(参考 5.1.1,5.4.8 节)。考察反序表,当 $N \leqslant P$ 时概率为 1,当 $N \geqslant P$ 时,概率为 $P^{N-P}P!/N!$。(然而在实际的做法中,一次扫描的排序并不太罕见,因为即使当人们怀疑一个文件是否有序时,作为一种预防措施,人们也倾向于把这个文件排序。)

17. 除开长度为 P 的最后者外,所有的都恰有$\lceil N/P \rceil$个路段("最坏的情况")。

18. 在第二次扫描时没有什么变化,因为能够证明,对于 $1 \leqslant k \leqslant P$,一个路段的第 k 个记录至少小于前一个路段的 $P+1-k$ 个记录。(然而,似乎没有简单的方法来表征当 $P' > P$ 时,P 路替代选择后边接以 P' 路替代选择的结果。)

19. 如同在(2)的推导中那样论证,$h(x,t)\mathrm{d}x = KL\mathrm{d}t$,这一次对于所有 x 有 $h(x,t) = 1 + Kt$,而且 $P = LI$。这意味着 $x(t) = L\ln((I+Kt)/I)$,使得当 $x(T) = L$ 时,我们有 $KT = (e-1)I$。故从 $t=0$ 起的落雪量是 $(e-1)LI = (e-1)P$。

20. 如同在习题 19 中那样,我们有 $(I + Kt)\mathrm{d}x = K(L - x)\mathrm{d}t$;因此 $x(t) = LKt/(I + Kt)$。水库中积雪量是 $LI = P = P' = \int_0^T x(t)K\mathrm{d}t = L(KT - I\ln((I + KT)/I))$;因此 $KT = \alpha I$,其中 $\alpha \approx 2.14619$ 是 $1 + \alpha = \mathrm{e}^{\alpha - 1}$ 的根。路段长度是在 $0 \leqslant t \leqslant T$ 期间下雪的总量,即 $LKT = \alpha P$。

21. 如同在正文中那样进行,但在每个路段之后,在扫雪机再次开始工作之前等候 $P - P'$ 个雪花落下。这意味着 $h(x(t),t)$ 现在是 KT_1 而不是 KT,其中 $T_1 - T$ 是额外的落雪所花费的时间数量。路段长度是 LKT_1,$x(t) = L(1 - \mathrm{e}^{-t/T_1})$,$P = LKT_1 e^{-T/T_1}$,以及 $P' = \int_0^T x(t)K\mathrm{d}t = P + LK(T - T_1)$。换言之,当对于 $0 \leqslant \theta \leqslant 1$,$P' = (1 - (1 - \theta)\mathrm{e}^{\theta})P$ 时,得到长度为 $\mathrm{e}^{\theta}P$ 的一个路段。

22. 对于 $0 \leqslant t \leqslant (\kappa - 1)T$,$\mathrm{d}x \cdot h = K\mathrm{d}t(x(t+T) - x(t))$,而对于 $(\kappa - 1)T \leqslant t \leqslant T$,$\mathrm{d}x \cdot h = K\mathrm{d}t(L - x(t))$,其中 h 在扫雪犁的位置处被看作恒等于 KT。由此得出,对于 $0 \leqslant j \leqslant k$,$0 \leqslant u \leqslant 1$,$t = (\kappa - j - u)T$,我们有 $x(t) = L(1 - \mathrm{e}^{u-\theta}F_j(u)/F(\kappa))$。路段长度是 LKT,这即是在稳定状态下扫雪机接连两次离开 0 点的时间之间下雪的数量;P 是在每次扫雪机的最后速度突变期间清除的数量,即 $KT(L - x(\kappa T)) = LKTe^{-\theta}/F(\kappa)$;而且可以证明 $P' = \int_0^{\kappa T} x(t)K\mathrm{d}t$ 有所述形式。

[注意:结果是,对于 $k = 0$,所述公式也正确。当 $k \geqslant 1$ 时,进入雪堆两次的每个路段的雪花个数是 $P'' = \int_0^{(\kappa-1)T} x(t)K\mathrm{d}t$,而且容易证明(路段长度)$- P' + P'' = (\mathrm{e} - 1)P$,这是由 Frazor 和黄泽权所指出的一个现象。$F_k(\theta)$ 的生成函数如此类似于习题 5.1.3-11 中的生成函数,这是否巧合呢?

23. 设 $P = pP'$ 且 $q = 1 - p$。在积雪中的最初的 pP' 个雪花已经以随机次序在开头移走之后,头 T_1 个时间单位的落雪来自积雪中剩下的 qP' 个雪花;而且当旧的

积雪扫光时,雪再次均匀落下。我们选择 T_1 使得 $LKT_1 = qP'$。对于 $0 \leqslant t \leqslant T_1$,$h(x,t) = (p + qt/T_1)g(x)$,其中 $g(x)$ 是雪从位置 x 落入雪堆的高度;对于 $T_1 \leqslant t \leqslant T$,$h(x,t) = g(x) + (t - T_1)K$。对于 $0 \leqslant t \leqslant T_1$,$g(x(t))$ 是 $(q(T_1 - t)/T_1)g(x(t)) + (T - T_1)K$;而对于 $T_1 \leqslant t \leqslant T$,$g(x(t)) = (T - t)K$。因此对于 $0 \leqslant t \leqslant T$,$h(x(t),t) = (T - T_1)K$,而且 $x(t) = L(1 - \exp(-t/(T - T_1)))$。总的路段长度是 $LK(T - T_1)$;从雪堆重新又"循环"回去的总数量是 LKT_1(参见习题 22);而且在时间 T 之后清除的总数量是 $P = KT(L - x(T))$。

所以当雪堆的大小为 $(1 + (s-1)e^s/s)P$ 时,这个习题的假定给出长度为 $(e^s/s)P$ 的路段。这比习题 22 的结果坏得多,因为雪堆的容量在习题 22 的情况下是以一个更有利的阶来使用的。

($h(x(t),t)$ 在这样多的问题当中都是常数这一事实是不足为怪的,因为它等价于说在系统的一个稳定状态期间,得到的每个路段的诸元素是一致分布的。)

24. (a) 证明实质上是相同的;每个子序列的路段和输出路段有相同的方向。(b) 所述的概率是路段的长度为 $n+1$ 而且它为 y 所跟随的概率;当 $x > y$ 时,它等于 $(1-x)^n/n!$,而当 $x \leqslant y$ 时,它是 $(1-x)^n/n! - (y-x)^n/n!$。(c) 归纳法。例如,如果第 n 个路段是递增的,则第 $n-1$ 个是递减的概率为 p,所以头一个积分适用。(d) 我们求得 $f'(x) = f(x) - c - pf(1-x) - qf(x)$,因此 $f''(x) = -2pc$,它最终导致 $f(x) = c(1 - qx - px^2)$,$c = 6/(3+p)$。(e) 如果 $p > eq$,则 $pe^x + qe^{1-x}$ 对于 $0 \leqslant x \leqslant 1$ 是单调递增的,而且 $\int_0^1 |pe^x + qe^{1-x} - e^{1/2}|\,dx = (p-q)(e^{1/2}-1)^2 < 0.43$。如果 $q \leqslant p < eq$,则 $pe^x + qe^{1-x}$ 处于 $2\sqrt{pqe}$ 和 $p + qe$ 之间,所以 $\int_0^1 |pe^x + qe^{1-x} - \frac{1}{2}(p + qe + 2\sqrt{pqe})|\,dx \leqslant \frac{1}{2}(\sqrt{p} - \sqrt{qe})^2 < 0.4$;而如果 $p < q$,则我们可以使用一个对称的推理。于是,对于所有的 p 和 q,有一个常数 C,使得 $\int_0^1 |pe^x + qe^{1-x} - C|\,dx < 0.43$。设 $\delta_n(x) = f_n(x) - f(x)$。则 $\delta_{n+1}(y) = (1 - e^{y-1})\int_0^1 (pe^x + qe^{1-x} - C)\delta_n(x)\,dx + p\int_0^{1-y} e^{y-1+x}\delta_n(x)\,dx + q\int_y^1 e^{y-x}\delta_n(x)\,dx$;因此如果 $\delta_n(y) \leqslant \alpha_n$,则 $|\delta_{n+1}(y)| \leqslant (1 - e^{y-1}) \cdot 1.43\alpha_n < 0.91\alpha_n$。(f) 对于所有 $n \geqslant 0$,$(1-x)^n/n!$ 是路段长度超过 n 的概率。(g) $\int_0^1 (pe^x + qe^{1-x})f(x)\,dx = 6/(3+p)$。

26. (a) 考虑具有 $n + r + 1$ 个元素和 n 个自左至右极小值的排列数,其中最右的元素不是最小的。(b) 借助于附录 B 中关于 Stirling 数的定义,利用

$$\sum_{1 \leqslant k < n} \left[{k \atop k - r} \right] k = \left[{n \atop n - r - 1} \right]$$

的事实。(c)把 $r+1$ 加到均值上,并且利用 $\sum_{n\geqslant 0}\left[{n+r \atop n}\right](n+r)\Big/(n+r+1)!=1$ 这一事实,得到 $\sum_{n\geqslant 0}\left[{n+r \atop n}\right]\Big/(n+r-1)!$。

(b)中的公式是由 P. Appell,*Archiv der Math. und Physik* **65** (1880),171～175 给出的。碰巧我们有 $\left[\left[{r \atop k}\right]\right]=(r+k)![x^k z^r]e^{xf(z)}$,其中 $f(z)=z2+z^2/3+\cdots=-z^{-1}\ln(1-z)-1$;因此 $c_r=[z^r](r+1+f(z))e^{f(z)}$,有 k 个循环的 n 个对象的去安排,有时以 $\left[{n \atop k}\right]_{\geqslant 2}$ 表示之,它等于 $\left[\left[{n-k \atop k}\right]\right]$;参见 J. Riordan,*An Introduction to Combinatorial Analysis* (Wiley,1958),§4.4。

27. 对于 $0\leqslant\theta\leqslant 1$,当 $P'/P=2(e^{-\theta}-1+\theta)/(1-2\theta+\theta^2+2\theta e^{-\theta})$时,稳定状态平均路段长度将是 $2P/(1-2\theta+\theta^2+2\theta e^{-\theta})$。[参见*Information Processing Letters* **21** (1985),239～243]。

Dobosiewicz 也发现,我们甚至可以更久地继续采用替代选择机制,因为我们可以从雪堆队的前端输入同时从它的后端输出。例如,如果 $P'=.5P$ 而且我们继续采用替代选择直到当前的路段包含 $.209P$ 个记录为止,则通过这个修改平均路段长度从大约 $2.55P$ 增加到大约 $2.61P$。如果 $P'=P$,而且我们继续采用替代选择直到在当前路段仅剩下 $.314P$ 个记录为止,平均路段长度从 eP 增加到大约 $3.034P$。[参见*Comp. J.* **27** (1984),334～339,其中还给出一个甚至更有效的称作"合并替代"的方法。]

28. 对于多路合并,问题比较小,因为 P 保持为常数,而且在每个文件上诸记录被顺序地处理;但当形成初始路段时,我们最好依照记录的长度来改变内存中记录的个数。利用如 2.5 节所述的动态分配策略,我们能保持装满整个内存那样多记录的一个堆。M. A. Goetz [*Proc. AFIPS Joint Computer Conf.* **25** (1964),602～604]已经提出了另一个方法,把每个记录分成为被链接在一起的固定大小的部分;它们在树叶处占有空间,但是仅仅前导部分参加比赛。

29. 顶层的 2^k 个失利者节点进到对应的宿主位置。剩下的失利者节点由每个有 2^n-1 个节点的 2^k 个子树组成;它们以对称的次序被指定到宿主节点中,即最左子树进到最左宿主节点去,等等。[参见 K. Efe 和 N. Eleser,*Acta. Informatica* **34** (1997),429～447。]

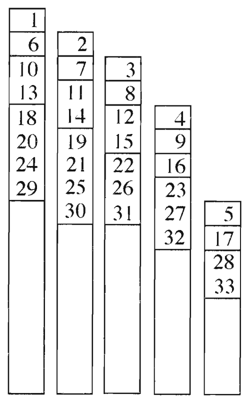
30. 假设宿主节点的 t 个保持完全的 2^{n+k} 节点失利者树的一个连通的 2^n 节点子图。对于 $1\leqslant l\leqslant n+k$ 该树在级 0 有一个节点而且在级 l 有 2^{l-1} 个节点。其根在 $l\geqslant 1$ 级处的一个子树有 $2^{n+k+1-l}-1$ 个节点;因此 t 个不相交的 2^n 个节点的子树的根必定全都在 $\leqslant k$ 的级上。而且这些子树的每一个在级 k 上必定至少含一个节点,因为在 $<k$ 的级上仅有 $2^{k-1}<2^n$ 个节点。由此得出,$t\leqslant 2^{k-1}$,但由(ii)和(iii),在宿主图中的边数至少是 $t+2(2^k-t)-1$,因为至少有这么多的失利者节点,

它们的父节点在这个宿主中有一个不同的映像。

[假设 $n \geqslant k$ 是必要的：当 $n=k-1$ 时，有一个含 $2^k+2^{k-1}-2$ 个边的适当的宿主图。]

5.4.2 小节

1.



2. 在头一个合并阶段之后，所有剩下的虚拟路段都在磁带 T 上，而且它们至多有 $a_n-a_{n-1} \leqslant a_{n-1}$ 个。因此在第二次合并阶段期间它们全都不出现。

3. 我们有(D[1],D[2],…,D[T]) $=(a_n-a_{n-P}, a_n-a_{n-P+1}, \cdots, a_n-a_n)$，所以这个条件从诸 a 是非减的这一事实得出。这一条件对于这个算法的正确性是重要的，因为在步骤 D2 和 D3 中 D[$j+1$]减值的次数决不比 D[j]减值的次数更多。

4. 由于(3)，$(1-z-\cdots-z^5)a(z)=1$。而且 $t(z)=\sum_{n\geqslant 1}(a_n+b_n+c_n+d_n+e_n)z^n=(z+\cdots+z^5)a(z)+(z+\cdots+z^4)a(z)+\cdots+za(z)=(5z+4z^2+3z^3+2z^4+z^5)\cdot a(z)$。

5. 设 $g_p(z)=(z-1)f_p(z)=z^{p+1}-2z^p+1$，而且设 $h_p(z)=z^{p+1}-2z^p$。Rouche 定理[*J. Ecole Polytechnique* **21**, **37** (1858), 1～34]告诉我们，倘若在圆上 $|h_p(z)|>|h_p(z)-g_p(z)|=1$，则 $h_p(z)$ 和 $g_p(z)$ 在圆 $|z|=1+\varepsilon$ 内有相同个数的根。如果 $\phi^{-1}>\varepsilon>0$，我们有 $|h_p(z)| \geqslant (1+\varepsilon)^p(1-\varepsilon)>(1+\phi^{-1})^2(1-\phi)^{-1}=1$。因此 g_p 有绝对值 $\leqslant 1$ 的 p 个根。它们是不同的，因为 $\gcd(g_p(z), g'_p(z))=\gcd(g_p(z), (p+1)z-2p)=1$。[*AMM* **67** (1960), 745～752。]

6. 设 $c_0=-\alpha p(\alpha^{-1})/q'(\alpha^{-1})$。则对于某个 $R>|\alpha|^{-1}$，$p(z)/q(z)-c_0/(1-\alpha z)$ 在 $|z| \leqslant R$ 中解析；因此在 $[z^n]p(z)/q(z)=c_0\alpha^n+O(R^{-n})$。于是，$\ln S=n \ln \alpha+\ln c_0+O(\alpha R)^{-n})$；而且 $n=(\ln S/\ln \alpha)+O(1)$ 蕴涵着 $O((\alpha R)^{-n})=O(S^{-\epsilon})$。类似地，设 $c_1=\alpha^2 p(\alpha^{-1})/q'(\alpha^{-1})^2$，$c_2=-\alpha p'(\alpha^{-1})/q'(\alpha^{-1})^2+\alpha p(\alpha^{-1})q''(\alpha^{-1})/q'(\alpha^{-1})^3$，并考虑 $p(z)/q(z)^2-c_1/(1-\alpha z)^2-c_2/(1-\alpha z)$。

7. 令 $\alpha_p=2x$ 和 $z=-1/2^{p+1}$。于是 $x^{p+1}=x^p+z$，所以由等式 1.2.6-25)，我们有收敛级数 $\alpha_p=2\sum_{k\geqslant 0}\binom{1-kp}{k}z^k/(1-kp)=2-2^{-p}-p2^{-2p-1}+O(p^2 2^{-3p})$。

注意：由此得出，当 p 增加时，习题 6 中的量 ρ 变成近似于 $\log_4 S$。类似地，在大量的磁带上，对于表 5 和表 6 两者，系数 c 趋于 $1/((\phi+2)\ln\phi)$。

8. 显然，对于 $m<0$，$N_0^{(p)}=1$，$N_m^{(p)}=0$，而且通过考虑对于头一个和式的不同可能性，当 $m>0$ 时，我们有 $N_m^{(p)}=N_{m-1}^{(p)}+\cdots+N_{m-p}^{(p)}$。因此 $N_m^{(p)}=F_{m+p-1}^{(p)}$。[*Lehrbuch der Combinatorik* :Teubner,1901),136～137。]

9. 如果有 1 的话，考虑最左边的那个 0 的位置，我们求得 $K_m^{(p)}=F_{m+p}^{(p)}$。

注意：在这样的 0 和 1 的序列和习题 8 中考虑的 $m+1$ 的表示之间，有一个简单的一一对应：把一个 0 放置在这个序列的右端，并观察所有 0 的位置。

10. 引理：如果 $n=F_{j_1}^{(p)}+\cdots+F_{j_m}^{(p)}$ 是这样一个表示，而且 $j_1>\cdots>j_m\geqslant p$，则我们有 $n<F_{j_1+1}^{(p)}$。证明：如果 $m<p$，则结论是明显的；否则设 k 是使 $j_k>j_{k+1}+1$ 的极小者；我们有 $k<p$，而且由归纳法 $F_{j_{k+1}}^{(p)}+\cdots+F_{j_m}^{(p)}<F_{j_{k-1}}^{(p)}$，因此 $n<F_{j_1}^{(p)}+\cdots+F_{j_1-k-1}^{(p)}\leqslant F_{j_1+1}^{(p)}$。

现在可对 n 用归纳法证明所述的结果。如果 $n>0$，则设 j 是使得 $F_j^{(p)}\leqslant n$ 的极大者。引理表明，n 的每个表示必须由 $F_j^{(p)}$ 加上 $n-F_j^{(p)}$ 的一个表示组成。由归纳法，$n-F_j^{(p)}$ 有所希望形式的一个惟一表示，而且这个表示不包括所有的数 $F_{j-1}^{(p)},\cdots,F_{j-p+1}^{(p)}$，因为 j 是极大者。

注意：在习题 1.2.8-34 中已经考虑了 $p=2$ 的情况，它是由 E. Zeckendovf 给出的[参见*Simon Stevin* **29** (1952),190～195]。从 n 的表示进到 $n+1$ 的表示，有一个简单的算法，它加工 0 和 1 的序列 $c_j\cdots c_1c_0$，使该序列满足 $n=\sum c_jF_{j+p}^{(p)}$：例如，如果 $p=3$，则我们考察右边的诸数字，把…0 变成为…1，…01 变成为…10，…011 变成为…100；然后如果需要，就进位到左边，以"…1000…"代替"…0111…"。(见习题 9 中按此次序列出的 0 和 1 的序列。)一个类似的数系，已为 W. C. Lynch 所研究[*Fibonacci Quarterly* **8** (1970),6～22]，他发现了一个非常有趣的方法，使它既支配一个多阶段排序的分布阶段，也支配合并阶段。

12. 第 k 次幂在它的诸行上逐次包含对于级 $k-4$ 到 k 的完全分布，最大的元素在右边。

13. 对级用归纳法。

14. (a) $n(1)=1$，所以假定 $k>1$。定律 $T_{nk}=T_{(n-1)(k-1)}+\cdots+T_{(n-P)(k-1)}$ 表明 $T_{nk}\leqslant T_{(n+1)k}$ 当且仅当 $T_{(n-P)(k-1)}\leqslant T_{n(k-1)}$。设 r 是任意正整数，并设 n' 是使 $T_{(n'-r)(k-1)}>T_{n'(k-1)}$ 之极小者；则对于所有 $n\geqslant n'$，$T_{(n-r)(k-1)}\geqslant T_{n(k-1)}$，因为对于 $n\geqslant n(k-1)+r$，这个关系是显然的，而且否则 $T_{(n-r)(k-1)}\geqslant T_{(n'-r)(k-1)}\geqslant T_{n'(k-1)}\geqslant T_{n(k-1)}$。(b)对于 $r=n-n'$ 的同样论证表明，$T_{n'k'}<T_{nk'}$ 蕴涵对于所有

$j \geqslant 0, T_{(n'-j)k'} \leqslant T_{(n-j)k'}$;因此这个递推式蕴涵对所有 $j \geqslant 0$ 和 $k \geqslant k'$, $T_{(n'-j)k} \leqslant T_{(n-j)k}$。(c) 设 $\ell(S)$是使得$\sum_n(S)$取它的极小值的最小的 n。当且仅当对所有 S, $\ell(S) \leqslant \ell(S+1)$时存在所希望的序列 M_n。假设 $n = \ell(S) > \ell(S+1) = n'$,使得 $\sum_n(S) < \sum_{n'}(S)$和$\sum_n(S+1) \geqslant \sum_{n'}(S+1)$。有某个最小的 S'使得$\sum_n(S') < \sum_{n'}(S')$,而且我们有 $m = \sum_n(S') - \sum_n(S'-1) < \sum_{n'}(S') - \sum_{n'}(S'-1) = m'$。所以$\sum_{k=1}^{m} T_{n'k} < S' \leqslant \sum_{k=1}^{m} T_{nk}$;因此有某个 $k' \leqslant m$,使得 $T_{n'k'} < T_{nk'}$。类似地,我们有 $l = \sum_n(S+1) - \sum_n(S) > \sum_{n'}(S+1) - \sum_{n'}(S) = l'$;因此$\sum_{k=1}^{l'} T_{n'k} \geqslant S+1 > \sum_{k=1}^{l'} T_{nk}$。由于 $l' \geqslant m' > m$,有某个 $k > m$,使得 $T_{n'k} > T_{nk}$。但这同(b)部分矛盾。

15. 这个定理已为 D. A. Zave 所证明,他的论文已在正文中引用。

16. D. A. Zave 已经证明,输入(和输出)记录的个数是 $S\log_{T-1}S + \frac{1}{2}S\log_{T-1}\log_{T-1}S + O(S)$。

17. 设 $T = 3$; $A_{11}(x) = 6x^6 + 35x^7 + 56x^8 + \cdots$, $B_{11}(x) = x^6 + 15x^7 + 35x^8 + \cdots$, $T_{11}(x) = 7x^6 + 50x^7 + 91x^8 + 64x^9 + 19x^{10} + 2x^{11}$。对于 $S = 144$ 的最优分布要求 T2 上的 55 个路段,而这使得 $S = 145$ 的分布不能是最优的。D. A. Zave 已经研究了这种类型的接近最优的过程。

18. 设 $S = 9$, $T = 3$,而且考虑下列两个型式

最优多阶段:

T1	T2	T3	费用
$0^2 1^6$	$0^2 1^3$	—	
1^3	—	$0^2 2^3$	6
—	$1^2 3^1$	2^2	5
3^2	3^1	—	6
3^1	—	6^1	6
—	9^1	—	9
			32

或者:

T1	T2	T3	费用
$0^1 1^6$	$0^1 1^3$	—	
1^3	—	$0^1 2^3$	6
—	$1^1 3^2$	2^1	7
3^1	3^2	—	3
—	3^1	6^1	6
9^1	—	—	9
			31

(还有另一个方法来改进"最优"的多阶段,这就是重新考虑在每个合并阶段的输出磁带上,应在哪里出现虚拟路段。例如,合并 $0^2 1^3$ 和 $0^2 1^3$ 的结果可以认为是 $2^1 0^1 2^1 0^1 2^1$,而不是 $0^2 2^3$。于是,仍然遗留着许多未解决的最优性问题。)

19.

级	T1	T2	T3	T4	总共	最后输出在
0	1	0	0	0	1	T1
1	0	1	1	1	3	T6

2	1	1	1	0	3	T5
3	1	2	1	1	5	T4
4	2	2	2	1	7	T3
5	2	4	3	2	11	T2
6	4	5	4	2	15	T1
7	5	8	6	4	23	T6
……						
n	a_n	b_n	c_n	d_n	t_n	T(k)
$n+1$	b_n	c_n+a_n	d_n+a_n	a_n	t_n+2a_n	$T(k-1)$

20. $a(z)=1/(1-z^2-z^3-z^4)$, $t(z)=(3z+3z^2+2z^3+z^4)/(1-z^2-z^3-z^4)$, $\sum_{n\geqslant 1}T_n(x)z^n=x(3z+3z^2+2z^3+z^4)/(1-x(z^2+z^3+z^4))$。$D_n=A_{n-1}+1$, $C_n=A_{n-1}A_{n-2}+1$, $B_n=A_{n-1}A_{n-2}A_{n-3}+1$, $A_n=A_{n-2}A_{n-3}A_{n-4}+1$。

21. 333343333332322 3333433333323 33334333333 3333433 333323 $T5$

22. $t_n-t_{n-1}-t_{n-2}=-1+3[n \bmod 3=1]$。(这个斐波那契式的关系是从下列事实得出的,即 $1-z^2-2z^3-z^4=(1-\phi z)(1-\hat{\phi}z)(1-\omega z)(1-\bar{\omega}z)$,其中 $\omega^3=1$)。

23. 在第 n 个合并阶段的头一半期间,路段长度不是(25),而是 s_n;在第二半为 t_n,其中

$$s_n=t_{n-2}+t_{n-3}+s_{n-3}+s_{n-4},\quad t_n=t_{n-2}+s_{n-2}+s_{n-3}+s_{n-4}$$

这里,我们认为对所有 $n\leqslant 0$, $s_n=t_n=1$。[一般地,如果 v_{n+1}是 $u_{n-1}+\cdots+v_{n-P}$的头 $2r$ 项的和,则我们有 $s_n=t_n=t_{n-2}+\cdots+t_{n-r}+2t_{n-r-1}+t_{n-r-2}+\cdots+t_{n-P}$;如果 v_{n+1}是头 $2r-1$ 项之和,则我们有 $s_n=t_{n-2}+\cdots+t_{n-r-1}+s_{n-r-1}+\cdots+s_{n-P}$, $t_n=t_{n-2}+\cdots+t_{n-r}+s_{n-r}+\cdots+s_{n-P}$]。

代替(27)和(28)的是,$A_n=(U_{n-1}V_{n-1}U_{n-2}V_{n-2}U_{n-3}V_{n-3}U_{n-4}V_{n-4})+1$, $\cdots$, $D_n=(U_{n-1}V_{n-1})+1$, $E_n=(U_{n-2}V_{n-2}U_{n-3})+1$; $V_{n+1}=(U_{n-1}V_{n-1}U_{n-2})+1$, $U_n=(V_{n-2}U_{n-3}V_{n-3}U_{n-4}V_{n-4})+1$。

25.

1^{16}	1^8	—	1^8
1^{12}	1^4	R	$1^8 2^4$
1^8	—	2^4	R

…………

R	$8^1 16^1$	8^1	8^0
16^0	R	8^1	—
16^1	16^1	8^0	R
R	16^1	—	24^0
16^1	16^1	R	$24^0 32^0$
16^0	16^0	32^1	(R)

26. 当 2^n 个路段被排序时，在合并时处理了 $n \cdot 2^n$ 个初始的路段；每一半的阶段(除少数例外)合并 2^{n-2}个和重绕 2^{n-1}个。当 $2^n + 2^{n-1}$个路段被排序时，在合并时处理了 $n \cdot 2^n + (n-1) \cdot 2^{n-1}$个初始的路段；每一半阶段(除少数例外)合并 2^{n-2}个或 2^{n-1}个并且重绕 $2^{n-1} + 2^{n-2}$个。

27. 当且仅当诸分布数的最大公因子是 1 时它有效。例如，设有六条磁带；如果我们分布(a,b,c,d,e)于 T1 到 T5 上，其中 $a \geqslant b \geqslant c \geqslant d \geqslant e > 0$，则头一阶段产生一个分布$(a-e, b-e, c-e, d-e, e)$，而且 $\gcd(a-e, b-e, c-e, d-e, e) = \gcd(a,b,c,d,e)$，因为一个数集的任何公因子也整除其它的公因子。这个过程减少每个阶段的路段数，直到 $\gcd(a,b,c,d,e)$个路段都在同一条磁带上为止。

[如同习题 18 中所示，在某些虚拟路段的配置下这些非多阶段的分布有时要比多阶段优越。这个现象是由 B. Sackman 大约于 1963 年首先发现的。]

28. 由(1,0,0,0,0)开始，并且进行下列操作恰好 n 次，我们即得任何这样的(a,b,c,d,e)：在$\{a,b,c,d,e\}$中选择 x，并把 x 加到(a,b,c,d,e)的其它四个元素中的每一个上。

为证明 $a+b+c+d+e \leqslant t_n$，我们将证明，如果在级 n 上，$a \geqslant b \geqslant c \geqslant d \geqslant e$，则总有 $a \leqslant a_n, b \leqslant b_n, c \leqslant c_n, d \leqslant d_n, e \leqslant e_n$。这个证明通过归纳法即可得出，因为级 $n+1$ 的诸分布是$(b+a, c+a, d+a, e+a, a)$，$(a+b, c+b, d+b, e+b, b)$，$(a+c, b+c, d+c, e+c, c)$，$(a+d, b+d, c+d, e+d, d)$，$(a+e, b+e, c+e, d+e, e)$。

30. 下表是由 J. A. Mortenson 计算出来的：

级	$T=5$	$T=6$	$T=7$	$T=8$	$T=9$	$T=10$	
1	2	2	2	2	2	2	M_1
2	4	5	6	7	8	9	M_2
3	4	5	6	7	8	9	M_3
4	8	8	10	12	14	16	M_4
5	10	14	18	17	20	23	M_5

6	18	20	26	27	32	31	M_6
7	26	32	46	47	56	42	M_7
8	44	53	74	82	92	92	M_8
9	68	83	122	111	138	139	M_9
10	112	134	206	140	177	196	M_{10}
11	178	197	317	324	208	241	M_{11}
12	290	350	401	488	595	288	M_{12}
13	466	566	933	640	838	860	M_{13}
14	756	917	1371	769	1064	1177	M_{14}
15	1220	1481	1762	2078	1258	1520	M_{15}
16	1976	2313	4060	2907	3839	1821	M_{16}

31.[*Random Structures & Algorithms* **5** (1994),102～104] $K_d(n)=F^d_{n-2}=N^{(d)}_{n-d-1}$。如果这棵树有 $r+1$ 个叶而且第 $k+1$ 个叶有这样 a_k-1 个祖先,它们是不同于头 k 个叶的祖先的,则我们有 $n-d-1=a_1+\cdots+a_r$。(七个例子树分别对应于 1+1+1+1,1+1+2,1+2+1,1+3,2+1+1,2+2 和 3+1。)

5.4.3 小节

1.当有 6,7 或 8 条磁带时,相对于每个记录被处理的平均次数来说,磁带分开多阶段是最优的(表 5.4.2-6)。

2.当初始路段数是一个斐波那契数时,这两个方法实际上是等同的;但是在其它情况下多阶段分布虚拟路段的方式更好些。级联算法把 1 放在 T1 上,然后放 1 于 T2 上,放 1 于 T1 上,放 2 于 T2 上,放 3 于 T1 上,放 5 于 T2 上,等等,而且当 $p=2$ 时,在步骤 C8 中决不会有 $D[p-1]=M[p-1]$。事实上,所有虚拟路段都在一条磁带上,而这是比算法 5.4.2D 中的方法更低效的。

3.(在步骤(3,3)期间在把 12 个路段放置在 $T3$ 上之后,分布停止。)

T1	T2	T3	T4	T5	T6
1^{26}	1^{21}	1^{24}	1^{14}	1^{15}	—
1^5	—	1^{12}	$1^2 2^7$	1^{15}	$2^2 4^{12}$
8^4	$6^2 9^3$	5^2	6^3	1^1	—
—	9^1	23^1	17^1	25^1	26^1
100^1	—	—	—	—	—

4.归纳法。(参见习题 5.4.2-28)。

5.当有 a_n 个初始路段时,第 k 趟扫描输出长度为 a_k 的 a_{n-k} 个路段,然后输出长度为 b_k 的 b_{n-k} 个路段,等等。

6. $\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$

7.我们节省 $e_2e_{n-2}+e_3e_{n-3}+\cdots+e_ne_0$ 个初始路段长度(参见习题 5),它也可以写成为 $a_1a_{n-3}+a_2a_{n-4}+\cdots+a_{n-2}a_0$;它是 $[z^{n-2}](A(z)^2-A(z))$。

8. $A(z)$ 的分母有不同的根和大于分子的次数,因此将 $A(z)=\sum q_3(\rho)/(1-\rho z)\cdot\rho(1-q_4'(\rho))$ 对于 $q_4(\rho)=\rho$ 的所有根求和。在计算 $q_3(\rho)$ 和 $q_4'(\rho)$ 中,ρ 的特殊形式是有帮助的。

9.鉴于 $q_m(2\sin\theta_k)$ 的值,根据(8)和(12),对所有大的 n 这些公式成立。为证明对于所有 n 它们都成立,我们需要知道对于 $0\leqslant m<r$,$q_{m-1}(z)$ 是当 $q_{r-1}(z)q_m(z)$ 除以 $q_r(z)-z$ 时的商。这可以通过以下几种方法来证明:或者使用(10)并注意相消减少了 $q_{r-1}(z)q_m(z)-q_r(z)q_{m-1}(z)$ 的次数,或者注意当 $z\to\infty$ 时,$A(z)^2+B(z)^2+\cdots+E(z)^2\to 0$(参见习题 5),或者对于 $B(z)$,$C(z)$ 等的分子,可以求得一个明显的公式。

10. $E(z)=r_1(z)A(z)$;$D(z)=r_2(z)A(z)-r_1(z)$;$C(z)=r_3(z)A(z)-r_2(z)$;$B(z)=r_4(z)A(z)-r_3(z)$;$A(z)=r_5(z)A(z)+1-r_4(z)$。于是 $A(z)=(1-r_4(z))/(1-r_5(z))$。[注意,$r_m(2\sin\theta)=\sin(2m\theta)/\cos\theta$;因此 $r_m(z)$ 是契比雪夫多项式 $(-1)^{m+1}U_{2m-1}(z/2)$]。

11.证明 $f_m(z)=q_{\lfloor m/2\rfloor}(z)-r_{\lceil m/2\rceil}(z)$ 和 $f_m(z)f_{m-1}(z)=1-r_m(z)$。然后用习题 10 的结果。(分母的这一显式首先是由 David. E. Ferguson 发现的。)

13.参见习题 5.4.6-6。

5.4.4 小节

1.当写出一个递增的路段时,在输出该路段之前首先写一个含有 $-\infty$ 的"哨兵"记录。(而如果这个输出将来是要被从后向前读的,如同在最后的扫描时那样,则 $+\infty$ 的哨兵也应写在这个路段的末端。)对于递减的路段,交换 $-\infty$ 和 $+\infty$ 的作用。

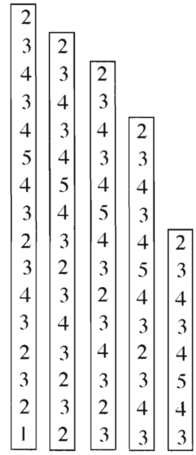
2.级 $n+1$ 上的最小数,等于级 n 上的最大数;因此不管在任何特定的行中我们排列诸数的方式如何,诸列是非减的。

3.事实上,在合并过程中,在 T2～T6 上的头一个路段将总是递减的,而在 T1 的头一个路段上将总是递增的。(由归纳法。)

4.在第二和第三阶段要求若干个"拷贝"操作;近似的额外费用是 $(\log 2)/(\log\rho)$ 次扫描,其中 ρ 是表 5.4.2-1 中的"增长率"。

5. 如果 α 是一个串，则令 α^R 表示它的左-右反向串。

级	T1	T2	T3	T4	T5
0	0	—	—	—	—
1	1	1	1	1	1
2	12	12	12	12	2
3	1232	1232	1232	232	32
4	12323432	12323432	2323432	323432	3432
…	…………	…………	…………	…………	…………
n	A_n	B_n	C_n	D_n	E_n
$n+1$	$B_n(A_n^R+1)$	$C_n(A_n^R+1)$	$D_n(A_n^R+1)$	$E_n(A_n^R+1)$	A_n^R+1



我们有

$$E_n = A_{n-1}^R + 1$$
$$D_n = A_{n-2}^R A_{n-1}^R + 1$$
$$C_n = A_{n-3}^R A_{n-2}^R A_{n-1}^R + 1$$
$$B_n = A_{n-4}^R A_{n-3}^R A_{n-2}^R A_{n-1}^R + 1$$

以及

$$A_n = A_{n-5}^R A_{n-4}^R A_{n-3}^R A_{n-2}^R A_{n-1}^R + 1 = n - Q_n$$

其中

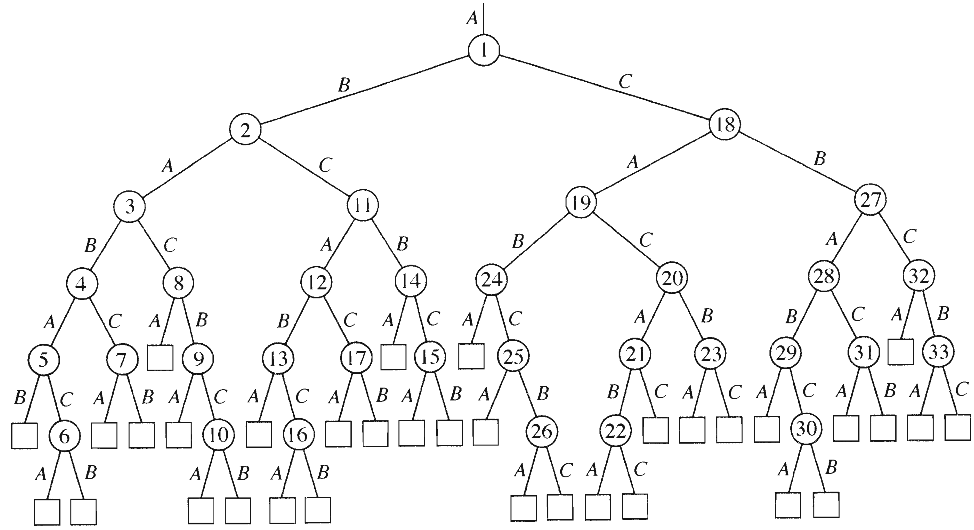
$$Q_n^R = Q_{n-1}(Q_{n-2}+1)(Q_{n-3}+2)(Q_{n-4}+3)(Q_{n-5}+4), \quad n \geqslant 1$$

$Q_0=0$，而且对于 $n<0, Q_n=\varepsilon$。

这些串 $A_n, B_n, \cdots$ 包含和 5.4.2 节中对应的串相同的项，但是按另一种次序。注意，相邻的合并数总差 1。一个初始路段必是 A 当且仅当它的合并数是偶数时，是 D 当且仅当它的合并数为奇数时。诸如算法 5.4.2D 那样的简单分布方案在把虚拟路段放置于高合并数位置中时，不是十分有效的；因此，在阶段 1 和阶段 2 之间计算 Q_n 大概是有利的，为的是帮助控制虚拟路段的设置。

6. $y^{(4)} = (+1, +1, -1, +1)$
$y^{(3)} = (+1, 0, -1, 0)$
$y^{(2)} = (+1, -1, +1, +1)$
$y^{(1)} = (-1, +1, +1, +1)$
$y^{0} = (\ 1,\ \ 0,\ \ 0,\ \ 0\ )$

7.（参见习题 15）顺便说一句，34 显然是这样一个最小的斐波那契数 F_n，对于它，多阶段排序不产生三条磁带上的 F_n 个初始路段的最优向后读合并。这株树的外部路径长度是 178，它胜过多阶段的 176。



8. 对于 $T=4$,具有外部路径长度 13 的树不是 T 后进先出的,而且具有外部路径长度 14 的每株树包括一个一路合并。

9. 按习题 2.3.4.5-6 的结果,我们可以考虑一株完备的$(T-1)$叉树;"最后"的内部节点的次数处于 2 和 $T-1$ 之间。当有$(T-1)^q-m$ 个外节点时,它们中的$\lfloor m/(T-2)\rfloor$个在级 $q-1$ 上,其余的则在级 q 上。

11. 对初始路段数用归纳法,可证其为真。如果存在一个具有 S 个路段的正确分布,而且有两个相邻的路段取同一方向,则有一个小于 S 个路段的分布;但当 $S=1$ 时不存在。

12. 条件(a)和(b)是明显的。如果对于某条磁带名 A 和某个 $i<j<k$,(4)中的两个配置之一存在,则由前根次序的定义,节点 j 必然是在节点 i 之下和节点 k 左边的一株子树中。因此"$j-l$"的情况不能存在,而且 A 必是"特殊"名,因为它出现在一支外部分支上。但这同这样一个事实矛盾,就是:特殊名应当在节点 i 之下最左边的分支上。

13. 现在编号为 4,7,11,13 的节点可以是外部的,而不是一路合并的。(这给出比多阶段树高 1 的外部路径长度。)

15. 设磁带名为 A,B 和 C。我们将构造若干种类的树,在植物学上这些树由它们的根和叶(外部节点)结构来进行标识:

类型 $r(A)$	根 A
类型 $s(A,C)$	根 A, 无 C 叶
类型 $t(A)$	根 A, 无 A 叶

类型 $u(A,C)$　根 A，无 C 叶，无复合 B 叶

类型 $v(A,C)$　根 A，无 C 叶，无复合 A 叶

类型 $w(A,C)$　根 A，无 A 叶，无复合 C 叶

一片"复合叶"是这样一片叶，它的兄弟不是一片叶，我们可以首先生成一株 $s(B,C)$类型的左子树，然后生成如 $r(C)$类型的右子树，来生成一个 3 后进先出类型的 $r(A)$树。类似地，类型 $s(A,C)$从类型 $s(B,C)$和 $t(C)$得出；类型 $u(A,C)$从类型 $v(B,C)$和 $w(C,B)$得出；类型 $v(A,C)$从类型 $u(B,C)$和 $w(C,A)$得出。我们可以生成一个 3 后进先出类型的 $t(A)$树，其左子树是 $u(B,A)$，其右子树是类型 $s(C,A)$，办法是首先让左子树生长，但它的(非复合) C 叶和它的右子树除外；而这时，左子树仅有 A 叶和 B 叶，所以我们可以生出整个树的右子树，然后长出左左子树的 A 叶，最后生成左右子树。类似地，一株 $w(A,C)$类型的树可以从一个 $u(B,A)$和一个 $v(C,A)$型的树构造出来。[习题 7 的树是一株以这样的方式构造的 $r(A)$树。]

令 $r(n),\cdots,w(n)$表示按上述过程构造出来的相应类型的所有 n 叶树的极小外部路径长度。我们有 $r(1)=s(1)=u(1)=0$，$r(2)=t(2)=w(2)=2$，$t(1)=v(1)=w(1)=s(2)=u(2)=v(2)=\infty$；而且对于 $n\geqslant 3$

$$r(n)=n+\min_k(s(k)+r(n-k)),\quad u(n)=n+\min_k(v(k)+w(n-k)),$$
$$s(n)=n+\min_k(s(k)+t(n-k)),\quad v(n)=n+\min_k(u(k)+w(n-k)),$$
$$t(n)=n+\min_k(u(k)+s(n-k)),\quad w(n)=n+\min_k(u(k)+v(n-k))$$

由此得出，对所有的 n，$r(n)\leqslant s(n)\leqslant u(n)$，$s(n)\leqslant v(n)$，以及 $r(n)\leqslant t(n)\leqslant w(n)$；进而，$s(2n)=t(2n+1)=\infty$。(后者是不证自明的。)

令 $A(n)$是由规则 $A(1)=0$，$A(2n)=2n+2A(n)$，$A(2n+1)=2n+1+A(n)+A(n+1)$定义的函数；则对所有 $n\geqslant 2$，$A(2n)=2n+A(n-1)+A(n+1)-(0\text{ 或 }1)$。设 C 是一个常数，使得对于 $4\leqslant n\leqslant 8$。

i) n 为偶数蕴涵 $w(n)\leqslant A(n)+Cn-1$。

ii) n 为奇数蕴涵 $u(n)$和 $v(n)\leqslant A(n)+Cn-1$。

(这实际上对于所有 $C\geqslant\frac{5}{6}$都有效。)则通过适当选择 k 为$\lfloor n/2\rfloor\pm 1$，归纳法论证表明，对于所有 $n\geqslant 4$，这些关系成立。但是 $A(n)$是当 $T=3$ 时(9)中的下限，且 $r(n)\leqslant\min(u(n),v(n),w(n))$，因此我们已经证明 $A(n)\leqslant\hat{K}_3(n)\leqslant r(n)\leqslant A(n)+\frac{5}{6}n-1$。[常数$\frac{5}{6}$还可以改进。]

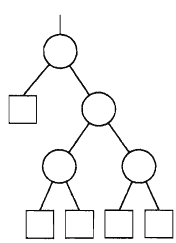
17. [下列方法首先用在 UNIVAC III 的排序程序中，并在 1962 年 ACM 排序讨论会上作了介绍。]

级	$T1$	$T2$	$T3$	$T4$	$T5$
0	1	0	0	0	0
1	5	4	3	2	1
2	55	50	41	29	15
……					
n	a_n	b_n	c_n	d_n	e_n
$n+1$	$5a_n+4b_n+3c_n+2d_n+e_n$	$4a_n+4b_n+3c_n+2d_n+e_n$	$3a_n+3b_n+3c_n+2d_n+e_n$	$2a_n+2b_n+2c_n+2d_n+e_n$	$a_n+b_n+c_n+d_n+e_n$

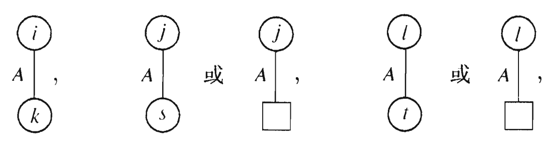
为了在初始分布期间从级 n 到达级 $n+1$,分别插入具有(4,4,3,2,1)个路段的 k_1 个"子级"加到带$(T_1,T_2,\cdots,T_5)$上,再加上具有(4,3,3,2,1)个路段的 k_2 个"子级",具有(3,3,2,2,1)个路段的 k_3 个"子级",具有(2,2,2,1,1)个路段的 k_4 个"子级",具有(1,1,1,1,0)个路段的 k_5 个"子级"上,其中 $k_1\leqslant a_n,k_2\leqslant b_n,k_3\leqslant c_n,k_4\leqslant d_n,k_5\leqslant e_n$。[如果$(k_1,\cdots,k_5)=(a_n,\cdots,e_n)$,则我们已经达到级 $n+1$。]必要时加虚拟路段以充满一个子级。然后从(T1,…,T5)合并 $k_1+k_2+k_3+k_4+k_5$ 个路段到 T6,从(T1,…,T4)合并 $k_1+\cdots+k_4$ 个到 T5,…,从 T1 合并 k_1 个到 T2;然后从(T2,…,T6)合并 k_1 个到 T1,从(T3,…,T6)合并 k_2 个到 T2,…,从 T6 合并 k_5 个到 T5。

18.(M.S. Paterson 给出的解。)假设把记录 j 写到磁带号为 τ_j 的序列上,至多有 $C_{|\tau|}$ 个记录有一个给定的序列,其中 C 依赖于内部存储的大小(见 5.4.8 节)。因此$|\tau_1|+\cdots+|\tau_N|=\Omega(N\log_T N)$。

19.



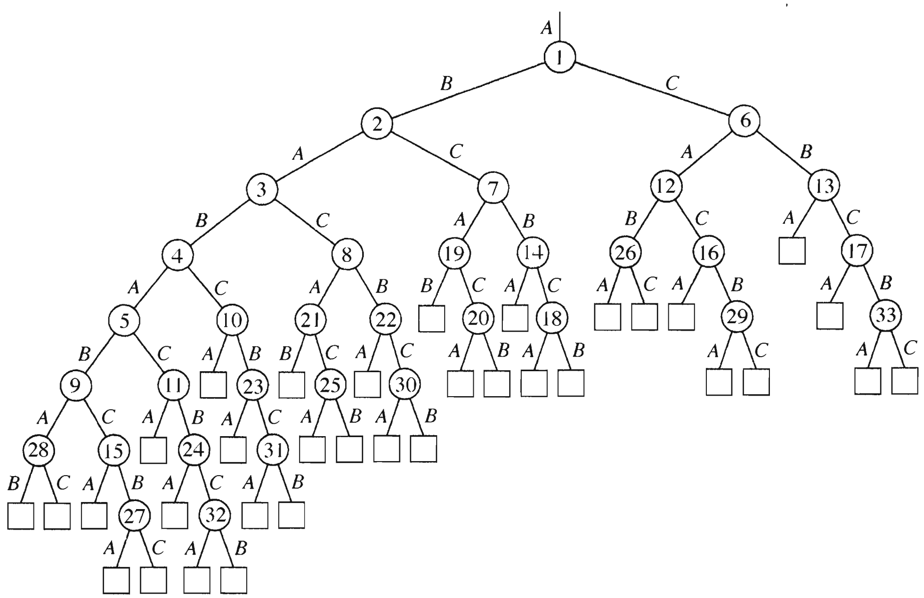
20.一个强 T 先进先出树有一组 T 先进先出标号,其中不存在分别具有如下形式



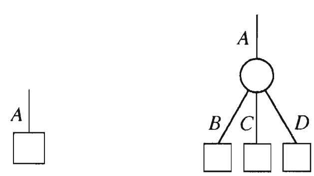
的三个分支,这里 A 是某个磁带名,且 $i<j<k<l<s$。非形式地说,当我们"长出"

一个 A 时,在建立任何新的 A 之前,必须长出所有其它的 A。

21.它是非常弱的先进先出:



22.对于通过例如,对于某些固定的磁带名 A,B,C,D 逐次地的所有出现,形成的任何树表示会出现这种情况。因为所有这些出现都为相同形式所代替,后进先出或先进先出次序在这个树结构中并不造成差别。



借助于向量模型叙述这个条件:每当($y^{(k+1)} \neq y^{(k)}$ 或 $k=m$)且 $y_j^{(k)}=-1$ 时,我们有 $y_j^{(k)}+\cdots+y_j^{(1)}+y_j^{(0)}=0$。

23.(a)假定 $v_1 \leqslant v_2 \leqslant \cdots \leqslant v_T$;"级联"阶段 $(1,\cdots,1,-1)^{v_T}(1,\cdots,1,-1,0)^{v_{T-1}}\cdots(1,-1,0,\cdots,0)^{v_2}$ 把 $C(v)$ 放入 v 中。(b)直接地,因为对于所有 k,$C(v)_k \leqslant C(w)_k$。(c)如果在 q 个阶段得到 v,则对于某个单位向量 u,以及某些其它向量 $u^{(1)},\cdots,u^{(q-1)}$ 我们有 $u \to u^{(1)} \to \cdots \to u^{(q)}=v$。因此 $u^{(1)} \leq C(u)$,$u^{(2)} \leq C(C(u)),\cdots,v \leq C^{[q]}(u)$。因此 $v_1+\cdots+v_T$ 小于或等于 $C^{[q]}(u)$ 的元素之和;而后者在级联合并中得到。[这个定理推广了习题 5.4.3-4 的结果。不幸,像这里所定义的"阶段"的概念,似乎没有任何实际的意义。]

24.设 $y^{(m)}\cdots y^{(l+1)}$ 是把 w 约化为 v 的一个阶段。如果对于某个 $k<i-1$,$y_j^{(i)}=-1$,$y_j^{(i-1)}=0,\cdots,y_j^{(k+1)}=0$,以及 $y_j^{(k)}=-1$,则我们可以插入 $y^{(k)}$ 于 $y^{(i)}$ 与

$y^{(i-1)}$之间。重复这个操作直到在每列中所有的(-1)都是相邻的为止。然后如果$y_j^{(i)}=0$和$y_j^{(i-1)}\neq 0$,则有可能置$y_j^{(i)}\leftarrow 1$;最后,每个列由一些$+1$,后边接上一些-1,再接上一些0组成,因此我们已经构造了一个阶段,对于某个$w'\geq w$,它把w'归结为v。这个阶段对各列进行排列,它具有形式$(1,\cdots,1,-1)^{a_T}\cdots(1,-1,0,\cdots,0)^{a_2}(-1,0,\cdots,0)^{a_1}$。$T-1$个关系的序列

$$\begin{aligned}(x_1,\cdots,x_T)\preceq(x_1+x_T,\cdots,x_{T-1}+x_T,0)\preceq\\ (x_1+x_{T-1}+x_T,\cdots,x_{T-2}+x_{T-1}+x_T,x_T,0)\preceq\\ (x_1+x_{T-2}+x_{T-1}+x_T,\cdots,x_{T-3}+x_{T-2}+x_{T-1}+x_T,x_{T-1}+\\ x_T,x_T,0)\preceq\cdots\preceq\\ (x_1+x_2+x_3+\cdots+x_T,x_3+\cdots+x_T,\cdots,x_{T-1}+x_T,x_T,0)\end{aligned}$$

表明诸a的最好选择是$a_T=v_T,a_{T-1}=v_{T-1},\cdots,a_2=v_2,a_1=0$。而且如果把诸列排列成$v_1\leqslant\cdots\leqslant v_T$,则结果是最好的。

25.(a)假设$v_{T-k+1}\geqslant\cdots\geqslant v_T\geqslant v_1\geqslant\cdots\geqslant v_{T-k}$,并使用$(1,\cdots,1,-1,0,\cdots,0)^{v_{T-k+1}}\cdots(1,\cdots,1,0,\cdots,0,-1)^{v_T}$。(b)对于$1\leqslant l\leqslant T-k$,$D_k(v)$的$l$个最大元素之和是$(l-1)s_k+s_{k+l}$。(c)如果在一个使用$k$个输出磁带的一个阶段中$v\Rightarrow w$,则我们显然可以假定该阶段有$(1,\cdots,1,-1,0,\cdots,0)^{a_1}\cdots(1,\cdots,1,0,\cdots,0,-1)^{a_k}$的形式,且其它$T-k$条磁带的每一条用作每个操作的输入。选择$a_1=v_{T-k+1},\cdots,a_k=v_T$是最好的。(d)见习题22(c)。我们总有$k_1=1$;而且$k=T-2$总是胜过$k=T-1$,因为我们假定,$v$的至少一个分量为0。因此对于$T=3$,有$k_1\cdots k_q=1^q$和初始分布$(F_{q+1},F_q,0)$。对于$T=4$,找到的不占优势的策略及其对应的分布是

$q=2$　12 (3,2,0,0)
$q=3$　121 (5,3,3,0) ; 122 (5,5,0,0)
$q=4$　1211 (8,8,5,0) ; 1222 (10,10,0,0) ; 1212(11,8,0,0)
$q=5$　12121 (19,11,11,0) ; 12222 (20,20,0,0) ; 12112(21,16,0,0)
$q=6$　122222 (40,40,0,0) ; 121212 (41,30,0,0)
$q\geqslant 7$　$12^{q-1}(5\cdot 2^{q-3},5\cdot 2^{q-3},0,0)$

故对于$T=4$和$q\geqslant 6$,极小阶段合并同平衡合并相似,只是在末尾处稍有曲折(从(3,2,0,0)进行到(1,0,1,1),而不是(0,0,2,1))。

当$T=5$时,不占优势的策略对于$q=2n\geqslant 2$是$1(32)^{n-1}2,1(32)^{n-1}3$;对于$q=2n+1\geqslant 3$是$1(32)^{n-1}32,1(32)^{n-1}22,1(32)^{n-1}23$。(所列的头一个策略在它的分布中有最多的路段。)在六条磁带上,它们是13或14,142或132或133,1333或1423,然后对于$q\geqslant 5$为13^{q-1}。

5.4.5 小节

1.下列算法为一个表A[$L-1$]…A[1]A[0]所控制,这个表实质上表示在基数P

记法下的一个数。当对这个数重复加 1 时,"进位"告诉我们何时合并。诸磁带被编号成从 $0 \sim P$。

O1.[初始化] 置(A[$L-1$],…,A[0])←(0,…,0)和 q←0。(在本算法期间,q 将等于(A[$L-1$]+…+A[0]) mod T)。

O2.[分布] 以递增次序写出磁带 q 上的一个初始路段。置 l←0。

O3.[加 1] 如果 $l=L$,则停止;以递增顺序输出在磁带($-L$) mod T 上,当且仅当 L 是偶数。否则置 A[l]←A[l]+1,q←(q+1) mod T。

O4.[进位?] 如果 A[l]<P,则返回 O2。否则合并到磁带($q-l$)mod T 上,置 A[l]←0 和 q←(q+1)mod T,l 增 1,并且返回 O3。 ▌

2.记住在每条磁带上有多少个路段。当输入穷尽时,必要时增加虚拟路段,并继续合并,直到达到在每条磁带上至多有一个路段和至少一条磁带是空的为止。然后在另一次合并中完成排序,必要时首先重绕某些磁带。(有可能从 A 表导出诸路段的方向。)

3.

OP	T0	T1	T2	OP	T0	T1	T2
分布	—	A_1	A_1A_1	分布	D_2A_1	A_1	A_4
合并	D_2	—	A_1	合并	D_2	—	A_4D_2
分布	D_2A_1	—	A_1	合并	—	A_4	A_4
合并	D_2	D_2	—	分布	—	A_4	A_4A_1
分布	D_2	D_2A_1	A_1	拷贝	—	A_4D_1	A_4
合并	D_2D_2	D_2	—	拷贝	—	A_4	A_4A_1
合并	D_2	—	A_4	合并	D_5	—	A_4

这时 T2 将被重绕,而最后的合并将完成此排序。

为了避免无用的拷贝操作,其中路段只是向前向后移动,我们可以在 B3 的结尾处说:"如果输入穷尽,则转到 B7",并且加上下列的新步骤:

B7.[做收尾工作]置 s←-1,并重复下列操作直到 $l=0$:置 s'←A[$l-1,q$],并置 q'和 r'成为使得 A[$l-1,q'$]$=-1$ 和 A[$l-1,r'$]$=-2$ 的下标,然后转到 B2。(对于 $j\neq q'$和 $j\neq r'$,我们将有 $q'=r$ 和 $s'\leqslant$A[$l-1,j$]$\leqslant s'+1$。)如果 $s'-s$ 为奇数,升高级 l,否则降低之(见下)。然后合并到磁带 r 上,并且向后读;置 l←$l-1$,A[l,q]←-1,A[l,r]←$s'+1$,r←r'并重复。这里"升高"指的是重复下列操作直到($q+(-1)^s$) mod $T=r$ 为止。置 p←($q+(-1)^s$)mod T 并且从磁带 p 拷贝一个路段到磁带 q,然后置 A[l,q]←$s+$

1, A$[l,p]\leftarrow -1$, $q\leftarrow p$。而“降低”指的是重复下列操作直到$(-q-(-1)^s) \bmod T=r$为止：置$p\leftarrow(q-(-1)^s) \bmod T$并从磁带$p$复制一个路段到磁带$q$，然后置A$[l,q]\leftarrow s$，A$[l,p]\leftarrow -1$，$q\leftarrow p$。拷贝操作在磁带$p$上向后读，因此它把被拷贝的路段的方向颠倒过来。当从p拷贝到q时，如果D$[p]>0$，我们简单地减少D$[p]$和增加D$[q]$而不作拷贝。

[基本的思想是，一旦穷尽了输入，则在每个磁带上我们要减少至多一个路段，每个非负项A$[l,j]$的奇偶性告诉我们一个路段是递增的还是递降的。这个变动造成差别的最小S是P^3+1。当P很大时，这个变动几乎不构成太多的差别，但它确定使计算机在某些情况下看上去很愚蠢。算法也应当加以改变以更有效地处理$S=1$的情况。]

4. 事实上，我们可以省略步骤B1中的设置A$[0,0]$，步骤B3和B5中的A$[l,q]$。[但是A$[l,r]$必须在步骤B3中设置。]在以前的答案中新的步骤B7确实需要A$[l,q]$的值。(除非如同在那里指出的那样，它明确地使用$q'=r$这个事实。)

5. 对于某个$k>0$，$P^{2k}-(P-1)P^{2k-2}<S\leqslant P^{2k}$。

5.4.6 小节

1. $\lfloor 23000480/(n+480)\rfloor n$。

2. 在所示时刻，该缓冲区中的所有记录都已经移至输出。步骤F2坚持，在合并时在测试“输入缓冲区是否空?”之前测试“输出缓冲区是否满?”，否则我们将会遇到麻烦(除非作了习题4的改变)。

3. 否；例如，如果对于$1\leqslant i\leqslant P$，文件i包含键码$i, i+P, i+2P, \cdots$，则我们可以达到P个缓冲区$1/P$满和$P-1$个缓冲区全满的状态。这个例子表明，为了连续进行输出，即使允许同时读，$2P$个输入缓冲区是必要的，除非我们重新分配内存作部分缓冲区来用某种方式使用“散列读”。[是的，实际上，如果诸块包含少于$P-1$个记录，则并不真正地需要$2P$个缓冲区；但这是不可能的。]

4. 早一点设置S。(在步骤F1和F4而不是F3。)

5. 例如，如果所有文件的所有键码都相等，则在预报时我们不能简单地作任意的决断；预报必须同合并处理所作的决断相容。一种安全的方式，是在步骤F1和F4中求最小的m，即每当$i<j$时认为取自文件C$[i]$的一个记录小于在文件C$[j]$上有相同键码的所有记录。(实质上，文件号被附加在键码上。)

6. 在步骤C1中，也置TAPE$[T+1]\leftarrow T+1$。在步骤C8中，应合并到TAPE$[p+2]$上而不是TAPE$[p+1]$上。在步骤C9中，置(TAPE$[1]$, …, TAPE$[T+1]$)←(TAPE$[T+1]$, …, TAPE$[1]$)。

7. 在图表A中使用的方法是$(A_1D_1)^4A_0D_0(A_1D_1)^2A_0D_0(A_1D_1)^3A_0$, $D_1(A_1D_1)^4A_0D_0(A_1D_1)^3A_0D_0\alpha A_0D_0A_0$, $D_1A_0D_0(A_1D_1)^3A_0D_0\alpha A_1D_1A_0$, $D_1A_1D_1\alpha A_1D_1A_0$，其中$\alpha=(A_0D_0)^2A_1D_1A_0D_0(A_1D_1)^2(A_0D_0)^7A_1D_1(A_0D_0)^3A_1D_1A_0D_0$。头一个合并阶段写$D_0A_3D_3A_1D_1A_4D_4A_0D_0A_1D_1A_1D_1A_4D_4A_0D_0$

$A_1D_1A_0D_0(A_1D_1)^4$于磁带5上；其次写$A_4D_4A_4D_4A_1D_1A_4D_4A_0D_0A_1D_1A_1D_1A_7$于磁带 1 上；其次写 $D_{13}A_4D_4A_0D_0A_{10}$于磁带 4 上。最后的阶段是

$A_4D_4A_4$	—	$D_{19}A_3D_3A_{12}$	$D_{13}A_4D_4A_4$	D_0A_3
A_4	$D_{23}A_{11}$	$D_{19}A_3$	$D_{13}A_4$	—
—	D_{23}	D_{19}	D_{13}	D_{22}
A_{77}	—	—	—	—

8.否。因为至多节约 S 次停止/启动，而且因为无论如何输入磁带(不是输出磁带)的速度总是要支配初始分布时间的。图表 A 中使用的分布方案的其它优点远远抵消了这微小的缺点。

9.$P=5,B=8300,B'=734,S=\lceil(3+1/P)N/(6P')\rceil+1=74,\omega\approx1.094,\alpha\approx0.795,\beta\approx-1.136,\alpha'=\beta'=0$；等式(9)≈855 s，我们对它们附加初始重绕的时间后总共为 958s。合并时间中节省大约 1 min，不足以补偿由于初始重绕和换磁带所损失的时间(除非也许我们处于一个多道程序设计环境中)。

10.在标准多阶段合并期间，重绕涉及该文件的大约 54%(表 5.4.2-1 中的“扫描/阶段”列)，而且由习题 5.4.3-5 和等式 5.4.3-(13)，在标准级联合并期间最长的重绕近似地包括文件的 $a_ka_{n-k}/a_n\approx(4/(2T-1))\cos^2(\pi/(4T-2))<\frac{4}{11}$。

11.仅仅初始和最后的重绕利用到“高速”的特征，因为当这个磁带卷包含整个举例文件时它仅仅比 10/23 满一点。利用例 8 中的 $\pi=\lceil .946\ln S-1.204\rceil,\pi'=1/8$，我们得到对于例 1～9 的下列估计的总数，分别为：

1115， 1296， 1241， 1008， 1014， 967， 891， 969， 856

12.(a) 使用 $4P+4$ 个缓冲区的一个显然的解决方法，就是简单地从成对的磁带同时读和写。但是注意，三个输出缓冲区是足够的：在一个给定的时刻，我们从一个缓冲区执行第二半写，从另一缓冲区执行头一半写，而输出到第三个缓冲区。而且这对于输入缓冲区状态提出了相应的改进。使用一项稍微弱化的“预报”技术可以证明，$3P$ 个输入缓冲区和 3 个输出缓冲区是必要的和充分的。肖智仁提出了一个更简单和更优越的方法，它把一个“向前看的键码”加到每一个块区中，以指明后继块的最后键码。肖智仁的方法要求 $2P+1$ 个输入缓冲区和 4 个输出缓冲区，因而这是对算法 F 的一个直截了当的修改。

(b) 在这种情况下，α 的很高的值意味着我们必须对数据进行五到六次扫描，它消除了双重快速合并的优点。这个思想在八条或九条磁带上实现时效果要好得多。

13.否，例如考虑正好在 $A_{16}A_{16}A_{16}A_{16}$之前的状态。但可处理两个满卷。

14.

$$\det\begin{pmatrix}0 & -p_0z & 0 & z-1\\ 0 & 1-p_1z & -p_0z & z-1\\ 1 & 0 & 0 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}\Big/\det\begin{pmatrix}1-p_{\geqslant 1}z & -p_0z & 0 & z-1\\ -p_{\geqslant 2}z & 1-p_1z & -p_0z & z-1\\ 0 & -1 & 1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}$$

15. A 矩阵有形式

$$A=\begin{pmatrix}B_{10}z & B_{11}z & \cdots & B_{1n}z & 1-z\\ \vdots & & & & \vdots\\ B_{n0}z & B_{n1}z & \cdots & B_{nn}z & 1-z\\ 0\cdots0 & 1 & 0 & & 0\\ 0\cdots0 & 0 & 0 & & 0\end{pmatrix},\quad \begin{matrix}B_{10}+B_{11}+\cdots+B_{1n}=1\\ \vdots\\ B_{n0}+B_{n1}+\cdots+B_{nn}=1\end{matrix} \tag{11}$$

因此

$$\det(I-A)=\det\begin{pmatrix}1-B_{10}z & -B_{11}z & \cdots & -B_{1(n-1)}z & -B_{1n}z\\ \vdots & & & & \vdots\\ -B_{n0}z & -B_{n1}z & \cdots & 1-B_{n(n-1)}z & -B_{nn}z\\ 0 & 0 & & -1 & 1\end{pmatrix}$$

而我们可以把所有列都加到头一列上，然后提出因子$(1-z)$。结果 $g_Q(z)$有$h_Q(z)/(1-z)$的形式，且 $\alpha^{(Q)}=h_Q(1)$，因为 $h_Q(1)\neq0$，且对于$|z|<1$，$\det(I-A)\neq0$。

5.4.7 小节

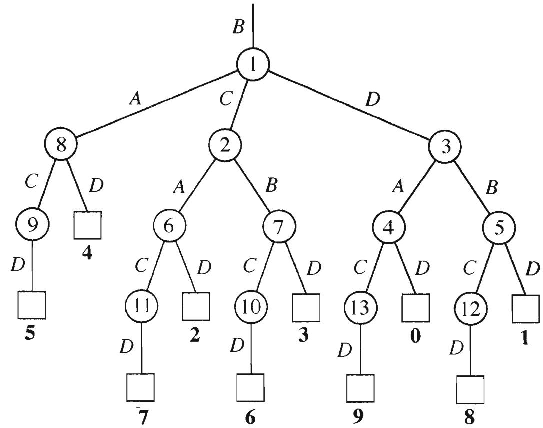
1. 在基数交替地为 P 和 $T-P$ 的数系中从最低位有效数字到最高位有效数字进行排序。(如果把数字对偶组合在一起，则我们实际上有 $P\cdot(T-P)$的一个纯基数。例如，如果 $P=2$ 和 $T=7$，则这个数系是以一种简单的方式同十进记号相关的"双五进制"。)

2. 如果 K 是 0 和 F_n-1 之间的一个键码，则设 F_n-1-K 的斐波那契表示是 $a_{n-2}F_{n-1}+\cdots+a_1F_2$，其中 a_j 为 0 或 1，而且不出现两个连续的 1。在阶段 j 之后，在 $a_{j-1}\cdots a_1$ 的递减次序下，磁带$(j+1) \bmod 3$ 包含具有 $a_j=0$ 的那些键码，而磁带$(j-1) \bmod 3$ 包含具有 $a_j=1$ 的那些键码。

[想像具有两个袋子"0"和"1"的一个卡片排序机，并且考虑 F_n 张已经在 $n-2$ 列穿上键码 $a_{n-2}\cdots a_1$ 的卡片的排序过程。把这些卡片排成递减次序的惯常步骤是从最低位有效数字开始，这可以简化，因为我们知道每次扫描末尾时在"1"袋中的每件事物下次扫描时将转入"0"袋中]。

4. 如果在级 2 上有一个外节点，则我们不能够构造这样一株好的树。否则，在级 3 上至多有三个外节点，在级 4 上六个，因为每个外节点应当都出现于相同的磁带上。

5.



6. 09, 08, …, 00, 19, …, 10, 29, …, 20, 39, …, 30, 40, 41, …, 49, 59, …, 50, 60, 61, …, 99。

7. 是;首先分布诸记录于越来越小的子文件上,直到得到一些单卷的文件为止,这些单卷的文件可以独立地排序。这对偶于下列过程:首先对诸单卷文件进行排序,而后把它们合并成越来越大的多卷文件。

5.4.8 小节

1. 是。如果在选择树中我们交替地使用递增和递减次序,则我们实际上有一个阶 P 的鸡尾混合排序。(参见习题 9。)

2. 设 $Z_N = Y_N - X_N$,并注意到 $(N+1)NZ_{N+1} = N(N-1)Z_n + N^2 + N$,即可解 Z_N 的这个递推式;因此

$$Z_N = \frac{1}{3}(N+1) + \binom{M+2}{3}\Big/ N(N-1), \quad \text{对于 } N > M$$

现在消去 Y_N 并得到

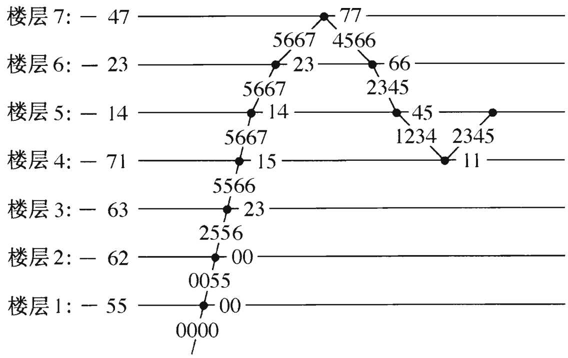
$$\frac{X_N}{N+1} = \frac{20}{3}(H_{N+1} - H_{M+2}) + 2\left(\frac{1}{N+1} - \frac{1}{M+2}\right) - \frac{2}{3}\binom{M+2}{3}\left(\frac{1}{(N+1)N(N-1)} - \frac{1}{(M+2)(M+1)M}\right) + \frac{3M+4}{M+2}, \quad N > M$$

3. 是;利用类似于定理 5.3.3L 的一个构造并用它来分划文件,即可在 $O(N)$ 步内求一个中间元素。由 R. W. Floyd 和 A. J. Smith 给出的另一个有趣的方法,是在 $O(N)$ 个时间单位内合并两个 N 项的路段如下:把诸项连同它们之间的空格,散布在诸磁带上,然后对每个空格逐个地用一个指明刚好居于该空格之前的项的最后位置的数,填入其中。

4. 有可能把对于层 $\{1, \cdots, p+1\}$ 的一个调度和对于层 $\{q, \cdots, n\}$ 的一个调度结合在一起:当前边的调度首先达到层 $p+1$ 时,向上到 q 层并进行后一个调度(利用

当前的电梯内容就好像它们是在定理 K 的算法中“额外的”个人那样)。在完成了该调度之后,回到层 $p+1$ 并且恢复以前的调度。

5. 考虑 $b=2, m=4$ 及算法的下列行为



现在 2(在电梯里)小于 3(在楼层 3 上)。

[在构造了这样一个例子之后,读者应该能看出怎样来揭示在定理 K 的证明中所需要的较弱的性质。]

6. 设 i, j 是使 $b_i < b'_i$ 和 $b_j > b'_j$ 的极小者。引进一个新的人,他要从 i 层到 j 层去。这对于任何 k 都不会增加 $\max(u_k, d_{k+1}, 1)$ 或 $\max(b_k, b^1_k)$。继续进行这过程直到对于所有 $j, b_j = b'_j$ 为止。现在注意若在步骤 K1 和 K3 中以 b_k 代替 b,则正文中的算法仍有效。

8. 设这个数是 P_n,且设 Q_n 是使得对于 $1 \leqslant k < n$, $u_k = 1$ 的排列数,则 $P_n = Q_1P_{n-1} + Q_2P_{n-2} + \cdots + Q_nP_0$, $P_0 = 1$。可以证明,对于 $n \geqslant 2$, $Q_n = 3^{n-2}$(见以下),因此用生成函数可推出

$$\sum P_nz^n = (1-3z)/(1-4z+2z^2) = 1 + z + 2z^2 + 6z^3 + 20z^4 + 68z^5 + \cdots$$

$$2P_n = (2+\sqrt{2})^{n-1} + (2-\sqrt{2})^{n-1}$$

为了证明 $Q_n = 3^{n-2}$,考虑三进序列 $x_1x_2\cdots x_n$,使得 $x_1 = 2, x_n = 0$ 且对于 $1 < k < n$,有 $0 \leqslant x_k \leqslant 2$。下列规则定义了在这样的序列和所希望的排列 $a_1a_2\cdots a_n$ 之间的一一对应:

$$a_k = \begin{cases} \max\{j \mid (j < k \text{ 且 } x_j = 0) \text{ 或 } j = 1\}, & \text{如果 } x_k = 0 \\ k, & \text{如果 } x_k = 1 \\ \min\{j \mid (j > k \text{ 且 } x_j = 2) \text{ 或 } j = n\}, & \text{如果 } x_k = 2 \end{cases}$$

(这个对应是由作者同 E. A. Bender 一起得到的。)

9. 鸡尾混合排序的扫描次数是 $2\max(u_1, \cdots, u_n) - (0 \text{ 或 } 1)$,因为每对扫描(左-右-左)都使得每个非 0 的 u 减 1。

10. 从一个分布方法(快速排序或基数交换)开始,直到得到一些单卷文件为止。而且要耐心点。

5.4.9 小节

1. $\frac{1}{4}-\left(x \bmod \frac{1}{2}\right)^2$ 圈转动。

2. 对于固定的 k,q,r 和 $i\neq i'$，$k=a_{i_q}$ 和 $k+1=a_{i'r}$ 的概率是 $f(q,r,k)L!L!\cdot(PL-2L)!/(PL)!$，其中

$$f(q,r,k)=\binom{k-1}{q-1}\binom{k-q}{r-1}\binom{PL-k-1}{L-q}\binom{PL-k-1-L+q}{L-r}=$$

$$\binom{k-1}{q+r-2}\binom{q+r-2}{q-1}\binom{PL-k-1}{2L-q-r}\binom{2L-q-r}{L-q}$$

而且

$$\sum_{\substack{1\leqslant k<PL\\1\leqslant q,r<L}}|q-r|f(q,r,k)=$$

$$\sum_{1\leqslant q,r\leqslant L}|q-r|\binom{PL-1}{2L-1}\binom{q+r-2}{q-1}\binom{2L-q-r}{L-q}=\binom{PL-1}{2L-1}A_{2L-1}$$

对于固定的 k,q 和 i，$k=a_{iq}$ 和 $k+1=a_{i(q+1)}$ 的概率是

$$g(k,q)\Big/\binom{PL}{L},\quad 其中\ g(k,q)=\binom{k-1}{q-1}\binom{PL-k-1}{L-q-1}$$

以及

$$\sum_{\substack{1\leqslant k<PL\\1\leqslant q<L}}g(k,q)=\sum_{1\leqslant q<L}\binom{PL-1}{L-1}=(L-1)\binom{PL-1}{L-1}$$

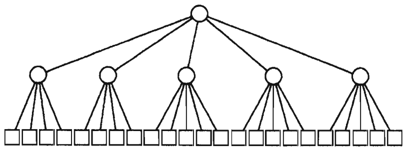
[*SICOMP* **1** (1972)，161～166。]

3. 在 $2\leqslant m\leqslant\min(9,n)$ 的范围内取(5)的极小值。

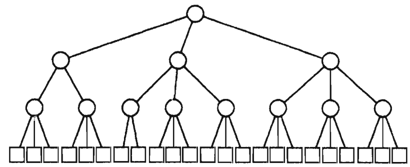
4. (a) $(0.000725(\sqrt{P}+1)^2+0.014)L$。(b)把公式(5)中的"$\alpha mn+\beta n$"改为"$(0.000725(\sqrt{m}+1)^2+0.014)n$"。[计算机的实验表明，通过这个新的递推式定义的最优树，和对于 $\alpha=0.00145$，$\beta=0.01545$ 由定理 K 所定义的那些最优树非常类似；事实上，当 $30\leqslant n\leqslant100$ 时，对于两个递推式都是最优的树是存在的。本题中提出的修改，如同正文中的例子那样，在 $n=64$ 或 100 时，节约大约 10% 的合并时间。这种风格的缓冲区分配已于 1954 年由 H. Seward 所考虑，他发现四路合并使寻找的时间极小化。]

5. 设 $A_m(n)$，$B_m(n)$ 表示其所有的 n 个叶分别在(偶，奇)级上的 m 株树的最优集合的费用。则 $A_1(1)=0$，$B_1(1)=\alpha+\beta$；当 $M\geqslant2$ 时像在(4)中那样定义 $A_m(n)$ 和 $B_m(n)$；$A_1(n)=\min_{1\leqslant m\leqslant n}(\alpha mn+\beta n+B_m(n))$，$B_1(n)=\min_{1\leqslant m\leqslant n}(\alpha mn+\beta n+A_m(n))$。后边的等式是正确定义的，虽则 $A_1(n)$ 和 $B_1(n)$ 是互相定义的！

6.



或



；

$A_1(23)=B_1(23)=268$。[奇怪，$n=23$ 是仅有的小于等于 50 的具下列性质的值，对于它，在奇偶性没有限制的情况下，任何具有 n 个叶的相等奇偶性的树都不是最优的。也许它是当 $\alpha=\beta$ 时仅有的这样的值。]

7. 考虑在任何树中的诸量 $\alpha d_1+\beta e_1,\cdots,\alpha d_n+\beta e_n$，其中 d_j 是对于第 j 片叶的次数之和，而 e_j 是路径长度。对于权 $w_1\leqslant\cdots\leqslant w_n$ 的一个最优树将有 $\alpha d_1+\beta e_1\geqslant\cdots\geqslant\alpha d_n+\beta e_n$。总是有可能重新对下标排序使得 $\alpha d_1+\beta e_1=\cdots=\alpha d_k+\beta e_k$，其中头 k 片叶被合并在一起。

9. 设 d 极小化 $(\alpha m+\beta)/\ln m$。使用凸性的一个简单归纳法证明 $A_1(n)\geqslant(\alpha d+\beta)n\log_d n$，而且当 $n=d^t$ 时等式成立。一个适当的上限从完备的 d 叉树得到，因为对于 $n=d^t+(d-1)r, 0\leqslant r\leqslant d^t$ 它们有 $D(\mathscr{T})=dE(\mathscr{T}), E(\mathscr{T})=tn+dr$。

10. 见*STOC* **6** (1974)，216～229。

11. 利用习题 1.2.4-38，当 $2\cdot3^{q-1}\leqslant n/m\leqslant3^q$ 时，$f_m(n)=3qn+2(n-3^qm)$；当 $3^q\leqslant n/m\leqslant2\cdot3^q$ 时，$f_m(n)=3qn+4(n-3^qm)$。于是 $f_2(n)+2n\geqslant f(n)$，而且当且仅当 $4\cdot3^{q-1}\leqslant n\leqslant2\cdot3^q$ 时等式成立；$f_3(n)+3n=f(n)$；$f_4(n)+4n\geqslant f(n)$，而且当且仅当 $n=4\cdot3^q$ 等式成立；对所有的 $m\geqslant5, f_m(n)+mn>f(n)$。

12. 利用描述一，1:1，1:1:1，1:1:1:1 或 2:2，2:3，2:2:2，…，$\lfloor n/3\rfloor:\lfloor(n+1)/3\rfloor:\lfloor(n+2)/3\rfloor$，…；对于 $4\cdot3^q\leqslant n\leqslant4\cdot3^{q+1}$这给出其所有叶在级 $q+2$ 上的树。(当$n=4\cdot3^q$时，形成两株这样的树。)

14. 穷举并考察 n 的所有分划，对于 $n=1,2,3\cdots$，可找到下列树的描述：－，1:1，1:1:1，1:1:1:1，1:1:1:1:1，1:1:1:1:1:1，1:1:1:1:3，1:1:3:3，3:3:3，1:3:3:3，3:4:4，3:3:3:3，3:3:3:4，3:3:4:4，3:4:4:4，4:4:4:4，…，5:6:6:6:12，6:6:6:6:12，6:6:6:6:13，…。(这些次数似乎总是$\leqslant$6，但这样一个结果看来十分难以证明。)

15. 如果 a 个人开始时上了电梯，则集中率在头一次停止时至多增加 $a+b$。当它下一次停止在最初一层时，比率至多增加 $b+m-a$。因此在 k 次停止之后，比率顶多增加 $kb+(k-1)m$。

16. 11 次停止：123456 至层 2，334466 至层 3，444666 至层 4，256666 至层 5，466666 至层 6，123445 至层 4，112335 至层 5，222333 至层 3，122225 至层 2，111555 至层 5，111111 至层 1。[这是极小的，因为由对称性，当电梯的容量任意时，停十次的一个解法，都可重新安排成依次停在层 2，3，4，5，6，p_2，p_3，p_4，p_5，1 上，其中

$p_2p_3p_4p_5$ 是{2,3,4,5}的一个排列;这样的调度只有当 $b \geqslant 8$ 时才可能。参考 Martin Gardner,*Knotted Doughnuts* (New York:Freeman,1986),第 10 章。]

17. 至少有 $(bn)!/b!^n$ 个配置;而且在 s 次停止之后由给定的一个停止可以得到的这个数至多是 $\left((n-1)\binom{b+m}{b}\right)^{s-1}$,由习题 1.2.6-67 它比 $(n((b+m)e/b)^b)^s$ 小。因此由习题 1.2.5-24 某些配置要求

$$s(\ln n + b(1+\ln(1+m/b))) > \ln(bn)! - nl_n b! >$$
$$bn \ln bn - bn - n((b+1)\ln b - b + 1)$$

注意:当 x 和 y 都为正时,使用 $1/(x+y) \geqslant \frac{1}{2}\min(1/x, 1/y)$ 这一事实,我们可以以下列方便的形式表述这个下限

$$\Omega\left(\min\left(nb, \frac{n\log(1+n)}{\log(1+m/b)}\right)\right)$$

A. Aggarwal 和 J. S. Vitter,*CACM* **31** (1988),1116～1127 已经得到相关的结果,他们还建立了匹配的上限

$$O\left(\min\left(nb, \frac{n\log(1+n)}{\log(1+m/b)}\right)\right)$$

关于对若干个磁盘的扩充,也参见 M. H. Nodine 和 J. S. Vitter,*ACM Symposium on Parallel Algor:thms and Architectures* **5** (1993),120～129。

18. 停止的预期数是 $\sum_{s\geqslant 1} p_s$,其中 p_s 是至少需要 s 次停止的概率。设 $q_s = 1 - p_{s+1}$ 是至多需要停止 s 次的概率。于是习题 17 表明 $q_s \leqslant f(s-1+[s=0])$,其中 $f(s) = b!^n\alpha^s/(bn)!$,以及 $\alpha = n((b+m)e/b)^b$,如果 $f(t-1) < 1 \leqslant f(t)$,则 $\sum_{s\geqslant 1} p_s \geqslant p_1 + \cdots + p_t = t - (q_0 + \cdots + q_{t-1}) \geqslant t - (f(0) + f(0) + \cdots + f(t-2)) \geqslant t - (\alpha^{1-t} + \alpha^{1-t} + \cdots + \alpha^{-1}) \geqslant t - 1 \geqslant L - 1$。

19. 考虑向后进行步骤(g),把诸记录分布到箱子 1 中,然后箱子 2。这个操作恰巧是步骤(iv)在键文件上模拟的操作。[*Princeton Conference on Information Sciences and Systems* **6** (1972),140～144。]

20. 内部排序必须小心地谨记分页的要求来选择;如果实际的内存很小,则诸如 Shell 排序、地址计算、堆排序以及表排序方法都将是灾难性的,因为它们要求大量页的"工作集合"。快速排序、基数交换以及顺序分配合并或基数排序是适合于一个分页环境的更好得多的方法。

一个外部排序的设计者能做的某些事情,实际上不可能包括在一个自动分页的方法中。这些事情是:(a)预报下一次应该读的输入文件,使得当需要时就有数据可用;(b)按照硬件和数据的特征选择缓冲区大小和合并的次序。

另一方面,如果程序员很仔细,而且知道所使用的实际机器的性质,则一部虚拟机器相当容易编程序,且它能给出不坏的结果。Brawn,Gustavson 和 Mankin 完成了关于这个问题最初的实质性的研究[*CACM* **13** (1970),483～494]。

21. $\lceil (L-j)/D \rceil$;参见*CMath* ,等式(3.24)。

22. 在阅读了包含 α_j 的一组 D 个块区之后，在阅读下一组 D 个块区之前，我们可能需要知道 α_{j+D-1}。而且如果我们以 a_j 来存储 α_{j+D-1}，我们也需要在某类文件标题中的值 $\alpha_0\cdots,\alpha_{D-2}$ 来使这个过程开始。

但是对于这个方案，在我们已经计算 $a_D\cdots a_{2D-2}$ 之前我们不能写块区 $a_0\cdots a_{D-1}$，所以我们将需要 $3D-1$ 个输出缓冲区而不是 $2D$ 个来使写保持连续。因此把诸 α 放在一个分开(短)的文件里更好。[对于随机分片同样的分析也适用。]

23. (a) 算法 5.4.6F 需要 4 个输入缓冲区，每个超块区的大小为 DB。(如果我们也计算输出的缓冲区，对算法 5.4.6F 我们总共有 $60DB$ 缓冲区记录在内存中，而对于 SyncSort(同步排序)有 $5DB$。)

(b) 当读一组 D 个块区时，我们需要供给以前的 D 个块区的缓冲空间，以及对于总共 $(2D+1)B$ 个记录的一个未完成的块区。(输出需要另外的 $2DB$，但是对于输入进行 2 路合并的许多数据操作产生比较少的输出。)

24. 设在年代次序下的第 l 个块区是路段 k_l 的块区 j_l；特别是，对于 $1\leqslant l\leqslant P$，$j_l=0$ 和 $k_l=l$。我们将在时间 $t_l=\sum_{k=1}^{P}t_{lkd}$ 时读该块区，其中

$$t_{lkd}=|\{r\mid 1\leqslant r\leqslant l \text{ 和 } k_r=k \text{ 以及 } (x_k+j_r)\bmod D=d\}|$$

是在按年代 $\leqslant l$ 的盘 d 上路段 k 的块区个数，而且 $d=(x_{kl}+j_l)\bmod D$，设 $U_{lk}=|\{r\mid 1\leqslant r\leqslant l \text{ 和 } k_r=k\}|$；于是

$$t_{lkd}=\left\lceil\frac{u_{lk}-(d-x_k)\bmod D}{D}\right\rceil$$

因为当 $1\leqslant r\leqslant l$ 和 $k_r=k$ 时 j_r 跑遍值 $0,1,\cdots,u_{lk}-1$。对于(19)，(20)和(21)的例子，序列 t_l 是

$$11111\quad 22223\quad 43456\quad 34567\quad 82345\quad 67893\cdots$$

如果 $l>P$，当我们从在编年次序下的第 l 个块区开始合并时需要的缓冲块区的个数是 I_l+D+P，其中 I_l 是 t_l 的"磁带有相等的反序"的个数，即 $|\{r\mid r>l \text{ 和 } t_r\leqslant t_l\}|$，这即是我们已经阅读但还未来得及使用的满缓冲区的个数；D 表示下一个输入所要用的缓冲区，P 表示部分满的缓冲区，我们而且正从中进行合并，(要特别小心，使用像在 SyncSort 中的链接时，我们可能把后一个要求从 P 减少成 $P-1$，但是额外的复杂性大概使这不值得。)

所以归结起来问题是得到对 I_l 的一个上限。我们可以假定，输入路段无穷地长。假设元素 $\{t_1,\cdots,t_l\}$ 中 s 个大于 t_l；则 t_l 有 t_lD-l+s 磁带有相等的反序，因为恰有 t_lD 个元素是 $\leqslant t_l$ 的。由此得出，当 $s=0$ 时极大值 I_l 出现，而且 t_l 是自左到右的极大值。我们有 $\sum_{k=1}^{P}u_{lk}=l$；因此由上面对于 t_l 的公式

$$I_l\leqslant\max_{l>P}(t_lD-l)\leqslant\sum_{k=1}^{P}(u_{lk}-(d-x_k)\bmod D+D-1-u_{lk})=$$

$$P(D-1)-\sum_{k=1}^{P}(d-x_k)\bmod D\leqslant$$

$$P(D-1)-\min_{0\leqslant d<D}\sum_{k=1}^{P}(d-x_k)\bmod D$$

因而存在编年次序，对于它们来说，可达到这个上限。

假设 x_k 中的 r_t 个等于 t，我们要选择 x_k 使得 $\min_{0\leqslant 1<D} sd$ 是极大的，其中 $s_d=\sum_{k=1}^{P}(d-x_k)\bmod D=\sum_{t=0}^{D-1}((d-t)\bmod D)r_t$。我们可以假定极小值在 $d=0$ 处出现。于是 $s_1=s_0+P-r_1D$，$s_2=s_1+P-r_2D$，…因此我们有 $r_1\leqslant\lfloor P/D\rfloor$，$r_1+r_2\leqslant\lfloor 2P/D\rfloor$，…由此得出，由习题 1.2.4-37，极小值是

$$s_0=(D-1)r_1+(D-2)r_2+\cdots+r_{D-1}\leqslant\sum_{k=1}^{D-1}\lfloor kP/D\rfloor=$$
$$\frac{1}{2}((P-1)(D-1)+\gcd(P,D)-1)$$

对于 $1\leqslant j\leqslant P$，当 $x_j=\lceil jD/P\rceil$时，达到这个上限。

如果我们有每个含 B 个记录的 $I_{\max}+D+P=\frac{1}{2}PD+\frac{3}{2}D+\frac{1}{2}P+\frac{1}{2}\gcd(P,D)-1$ 个输入缓冲区，通过这样的 x_j，我们可以以全速来处理每个编年次序的序列。（当 $D=2$ 或 3 时，这十分之好。）

25. 注意在时间 4 时，我们回头来读磁盘 0 的 f_1

	活动的阅读	活动的合并	草稿	等待
时间 1	$e_0b_0g_0a_0c_0$	$--------$	$(--------)$	a_0
时间 2	$f_1d_0d_1d_2f_0$	$a_0-------$	$b_0c_0(e_0g_0---)$	d_0
时间 3	$a_2h_0e_2g_1d_3$	$a_0b_0c_0d_0---$	$e_0f_0g_0(d_1d_2f_1-)$	h_0
时间 4	$f_1e_1b_1g_1a_1$	$a_0b_0c_0d_0e_0f_0g_0h_0$	$d_1(d_2e_2d_3f_1g_1a_2)$	e_1
时间 5	$a_2f_2h_1e_3g_2$	$a_0b_0c_0d_1e_1f_0g_0h_0$	$d_2e_2d_3a_1f_1b_1g_1l()$	a_2
时间 6	$d_4a_3f_3b_2e_4$	$a_2b_1c_0d_3e_2f_1g_1h_0$	$f_2e_3(h_1g_2---)$	d_4
时间 7	$c_1a_3f_3?\ e_4$	$a_2b_1c_0d_4e_3f_2g_1h_0$	$(h_1b_2g_2a_3f_3e_4-)$	c_1
时间 8	$?\ d_5d_6??$	$a_2b_1c_1d_4e_3f_2g_1h_0$	$h_1b_2g_2a_3f_3e_4(?)$	d_5

26. 当正在读 D 个块区和正在写 D 个块区时，在(24)的假定之下，合并过程可能生成多达 $P+Q-1$ 个输出的块区。（不是 $P+Q$，因为只有一个合并缓冲区变成完全空的。）读和写一样地快，所以对防止输出障碍来说 $D+P+Q-1$ 个输出缓冲是必要和充分的。

然而，平均说来，对于每 D 个输入块区，至多有 D 个块区是供输出的，所以在实用中 $3D$ 个输出缓冲区将是足够的。

27. (a) $E_n(m_1,\cdots,m_p)=\sum_{t=1}^{m}q_t$，其中 q_t 是某个瓮至少包含 t 个球的概率。显然，$q_t\leqslant 1$ 且

$$q_t \leqslant \sum_{k=0}^{n-1} \Pr(\text{瓮 } k \text{ 至少包含 } t \text{ 个球}) = n\Pr(S_n(m_1,\cdots,m_p) \geqslant t)$$

(b) s_n 的概率生成函数是

$$p(z) = \prod_{k=1}^{p} z^{qk}(1 + (z-1)r_k/n)$$

其中 $q_k = \lfloor m_k/n \rfloor$和 $r_k = m_k \bmod n$,现在当 $\alpha \geqslant 0$ 时,$1+\alpha \leqslant (1+\alpha/n)^n$ 和 $1+\alpha r/n \leqslant (1+\alpha/n)^r$;因此我们有 $\Pr(Sn(m_1,\cdots,m_p) \geqslant t) \leqslant (1+\alpha)^{-t}p(1+\alpha) \leqslant (1+\alpha)^{-t}\prod_{k=1}^{p}(1+\alpha/n)^{m_k} = (1+\alpha)^{-t}(1+\alpha/n)^m$。

如果 $t \leqslant m/n$,在所述的极小中我们用"1"的项。如果 $t > m/n$,当 $\alpha = (nt - m)/(m-t)$ 时量 $(1+\alpha)^{-t}(1+\alpha/n)^m$ 取它的极小值 $(n-1)^{m-t}m^m/(n^m t^t(m-t)^{m-t})$。

28. 数值的证据看来支持这个自然的猜测。例如:我们有

$E_{10}(1,1,1,1,1,1,1,1) = 2.3993180$, $E_{10}(2,2,2,2) = 2.178$, $E_{10}(4,3,1) = 2.00$,
$E_{10}(2,1,1,1,1,1,1) = 2.364540$, $E_{10}(3,2,2,1) = 2.166$, $E_{10}(5,2,1) = 1.98$,
$E_{10}(2,2,1,1,1,1) = 2.32076$, $E_{10}(3,3,1,1) = 2.152$, $E_{10}(6,1,1) = 1.94$,
$E_{10}(3,1,1,1,1,1) = 2.29958$, $E_{10}(4,2,1,1) = 2.138$, $E_{10}(4,4) = 1.7$,
$E_{10}(2,2,2,1,1) = 2.2628$, $E_{10}(5,1,1,1) = 2.090$, $E_{10}(5,3) = 1.7$,
$E_{10}(3,2,1,1,1) = 2.2460$, $E_{10}(3,3,2) = 2.02$, $E_{10}(6,2) = 1.7$,
$E_{10}(4,1,1,1,1) = 2.2076$, $E_{10}(4,2,2) = 2.01$, $E_{10}(7,1) = 1.7$,

29. (a) 在时间 t 时,所有磁盘所读的那些块区,都是不早于时间 t 时标记的块区才出现的。一旦它们已被阅读,下 Q 个块区决不从草稿缓冲区删去。因此在磁盘 j 上有关的块区都在小于等于 $t+N_j$ 的时间被阅读;在 $t+\max(N_0,\cdots,N_{D-1})$ 的时间它们必定全都参与到合并当中。

(b) 在一个加标记的块区之后如果第 $Q+1$ 个块区未被删去,则相同的论证适用。否则以前的 Q 个块区未被标记,因而 $Q+2$ 个块区不可能全都在不同的磁盘上。

(c) 把编年顺序的块区分成为大小 $Q+2$ 的一些组,并且考虑任何特定的组。如果从路段 k 有 M_k 个块区,则数 N_j 在对于 $n=D$ 和 $m=Q+2$ 的一个循环占据问题中,等价于第 j 个瓮中球的个数。因此在任何组中预期的标记了的单元数至多是习题 27(b) 中的上限。把这上限叫做 $e_n(m)$,我们可以取 $r(d,m) = (d/m)e_d\cdot(m)$。

[实际上,当 m 很小时,这个函数 $r(2,m)$ 不是对 m 单调的。因此对于表 2 中的 $r(2,4)$ 和 $r(2,12)$ 所列的项实际上是 $r(2,3)$ 和 $r(2,11)$ 的值;另外的缓冲区不可能增加被标记的块区的个数。]

30. 令 $l = \lceil (s + \sqrt{2s}\ln d \rceil$,$\alpha = \sqrt{2/s}$。于是

$$e_d(sd\ln d) < l + \sum_{t>1} d(1+\alpha/d)^{sd\ln d}/(1+\alpha)^t =$$

$$l + d(1 + \alpha/d)^{sd\ \ln d}/\alpha(1 + \alpha)^{l} \leqslant$$
$$l + \alpha^{-1}\exp((\ln d)(1 + s\alpha - (s + \sqrt{2s})\ln(1 + \alpha)))$$

而且$(s + \sqrt{2s})\ln(1 + \alpha) > s\alpha + 1 - \alpha/3$。因此如果 $s/(\log d)^2 \to \infty$,

$$1 \leqslant r(d, sd\ \ln d) = \frac{e_d(sd\ \ln d)}{s\ \ln d} < 1 + \sqrt{\frac{2}{s}} + \frac{1}{\sqrt{2s}\ln d}\left(1 + \sqrt{\frac{2}{9s}}\ln d + O(s^{-1}(\log d)^2)\right)$$

对这个渐近特性的收敛是稍微慢的(见表 2)。

31.(当 $Q=0$ 时,我们标记头一个块区而后重复地标记下一个这样的块区,在由以前已标记的块区开始的组中,这个块区同其中的一个块区共享一个磁盘。例如,如果磁盘的访问的编年次序是 112020121210122,则标记将是 $\bar{1}\ \bar{1}\ 2\ 0\ \bar{2}\ 0\ 1\ \bar{2}\ 1\ \bar{2}\ 1\ 0\ \bar{1}\ 2\ \bar{2}$。因此,当 $p \to \infty$时,在 n 个时间单位期间,我们平均读 $Q(D)n$ 个块区,其中 Q 是在等式 1.2.11.3-(2)中定义的 Ramanujan 函数。与之对照,$r(d,2) = (d+1)/2$ 给出一个悲观得多的估计。)

5.5 节

1.*在一个给定的状态下判定哪一个排序算法最好是困难的。* ▌

2.对于小的 N,表插入;对于中等的 N,例如,$N=64$,表合并;对于大的 N,基数表排序。

3.(由 V. Pratt 给出的解) 给定有待合并的两个非减的路段 α, β,以一个直截了当的方式确定子路段 $\alpha_1\alpha_2\alpha_3\beta_1\beta_2\beta_3$,使得 α_2 和 β_2 正好包含 α 和 β 的键码中具有整个文件中值的那些键码。通过逐次的"颠倒",首先形成 $\alpha_1\alpha_2\beta_1^R\alpha_3^R\beta_2\beta_3$,然后 $\alpha_1\beta_1\alpha_2^R\beta_2^R\alpha_3\beta_3$,然后 $\alpha_1\beta_1\alpha_2\beta_2\alpha_3\beta_3$,我们就能把这个问题归结为合并有长度$\leqslant N/2$ 的两个子文件 $\alpha_1\beta_1$ 和 $\alpha_3\beta_3$ 的问题。

由 L. Trabb Pardo 提出的一个相当复杂的算法提供了对于这一问题最好的渐近答案:仅仅使用 $O(\log N)$个二进位的辅助存储作为固定数目的下标变量之用,而无需以任何方式变换诸记录,我们就能在 $O(N)$的时间进行稳定的合并和以 $O(N\log N)$的时间进行稳定的排序。[*SICOMP* **6** (1977), 351 ~ 372)。黄秉超和 M. A. Langston, *Comp. J.* **35** (1992), 643～650 已经以小得多的常数因子实现相同的时间和空间的上限。当 M 比 N 小得多时,M 个项目和 N 个项目的稳定合并,也见 A. Symvonis, *Comp. J.* **38** (1995), 681～690。]

4.仅仅直接插入,表插入,以及表合并。快速排序的变形可以做成吝啬的,但是仅仅以内循环的额外工作为代价。(参见习题 5.2.2-24。)

当比较的结果不是 100% 可靠时,吝啬的方法特别有用,参见 D. E. Knuth, *Lecture Notes in Comp. Sci.* **606** (1992), 61～67。

6.1 节

1. $\sqrt{(N^2-1)/12}$。参见等式 1.2.10-(22)。

2. **S1′.** [初始化] 置 P←FIRST。

S2′. [比较] 如果 K = KEY(P)，则算法成功地结束。

S3′. [前进] 置 P←LINK(P)。

S4′. [文件结束?] 如果 P≠Λ，则转回 S2′。否则这个算法以失败告终。 ▌

3.
```
KEY      EQU   3:5
LINK     EQU   1:2
START    LDA   K             1
         LD1   FIRST         1
2H       CMPA  0,1(KEY)      C
         JE    SUCCESS       C
         LD1   0,1(LINK)     C-S
         J1NZ  2B            C-S
FAILURE  EQU   *             1-S    ▌
```

运行时间是 $(6C-3S+4)u$。

4. 是，如果我们有方法置“KEY(Λ)”等于 K 的话。[但在程序 Q′中使用的循环重复技术，在这种情况下无效。]

5. 否；程序 Q 总是至少做和程序 Q′同样多的操作。

6. 以 `JE *+4;CMPA KEY+N+2,1;JNE 3B;INC1 1` 代替行 08；而且把行 03～04 改变成为 `ENT1 -2-N; 3H INC1 3`。

7. 注意 $\bar{C}_N=\frac{1}{2}\bar{C}_{N-1}+1$。

8. 欧拉求和公式给出

$$H_n^{(x)}=\zeta(x)+\frac{n^{1-x}}{(1-x)}+\frac{1}{2}n^{-x}-\frac{B_2x}{2!}n^{-1-x}+\frac{B_3x(x+1)}{3!}n^{-2-x}-O(n^{-3-x})$$

复变理论告诉我们

$$\zeta(x)=2^x\pi^{x-1}\sin\left(\frac{1}{2}\pi x\right)\Gamma(1-x)\zeta(1-x)$$

这是当 $x<0$ 时特别有用的一个公式。

9. (a) 是：$\bar{C}_N=N-N^{-\theta}H_{N-1}^{(-\theta)}=N+1-N^{-\theta}H_N^{(-\theta)}=\frac{\theta}{1+\theta}N+\frac{1}{2}+O(N^{-\theta})$。

(b) $\bar{C}_N=\frac{\theta}{1+\theta}\left(1+N\Big/\left(1-\binom{N-\theta}{N}\right)\right)=\frac{\theta}{1+\theta}(N+N^{1-\theta}/\Gamma(1-\theta)+1)+$

$O(N^{1-2\theta})$。

(c) 当 $\theta<0$ 时,(11)不是一个概率分布;(16)给出 $\overline{C}_N=-\frac{\theta}{1+\theta}\Gamma(1-\theta)N^{1+\theta}+O(N^{1+2\theta})+O(1)$ 的估计以代替(15)。

10. $p_1\leqslant\cdots\leqslant p_N$;(极大值 $\overline{C}_N$) $=(N+1)-$(极小值 $\overline{C}_N$)。[类似地,在不等长情况下,极大平均查找时间是 $L_1(1+p_1)+\cdots+L_N(1+p_N)$ 减去极小平均查找时间。]

11. (a) $f_{m-1}(x_{i_1};\cdots,x_{i_{m-1}})p_i$ 的诸项恰好是可能已在前面的的诸要求序列的概率,这些要求序列均使 R_i 留在位置 m 中。(b) 第二个恒等式从累计在 X 的不同的 m 子集上开头的 $\binom{n}{m}$ 种情况得出,注意每个 P_{nk} 出现的次数。第三个恒等式从第二个通过反序得来。[或者,可以使用容斥原理。] (c) $\sum_{m\geqslant 0}mP_{nm}=nQ_{nn}-Q_{n(n-1)}$;因此

$$d_i=1+(N-1)-p_i\sum_{j\neq i}\frac{1}{p_i+p_j}$$

$$\sum p_id_i=N-\sum_{i<j}\frac{p_i^2+p_j^2}{p_i+p_j}=N-\sum_{i<j}\left(p_i+p_j-\frac{2p_ip_j}{p_i+p_j}\right)=$$

等式(17)

注意:W. J. Hendricks[*J. Applied Probability* **9** (1972),231~233]发现了对于记录的每一个排列的稳定状态概率的一个简单公式。例如,当 $N=4$ 时,这个序列将是 $R_3R_1R_4R_2$ 且有极限概率

$$\frac{p_3}{p_3+p_1+p_4+p_2}\quad\frac{p_1}{p_1+p_4+p_2}\quad\frac{p_4}{P_4+p_2}\quad\frac{p_2}{p_2}$$

James Bitner[*SICOMP* **8** (1979),82~85]证明,如果表原来处于随机次序,则在 t 次随机请求之后预期的查找时间超过 $\widetilde{C}_N$ 的数量是 $\frac{1}{4}\sum_{i,j}(p_i-p_j)^2(1-p_i-p_j)^t/(p_i+p_j)$。因此平均说来,$t$ 次查找加在一起要求少于 $t\widetilde{C}_N+\frac{1}{4}\sum_{i,j}(p_i-p_j)^2/(p_i+p_j)^2<t\widetilde{C}_N+\frac{1}{2}\binom{N}{2}$ 次比较。关于通过生成函数给出的有启发的证明,请见 P. Flajolet, D. Gardy 及 L. Thimonier, *Discrete Applied Math.* **39** (1992),207~229, §6。

12. $\widetilde{C}_N=2^{1-N}+2\sum_{n=0}^{N-2}1/(2^n+1)$,它迅速地收敛到 $2\alpha'\approx2.5290$;习题 5.2.4-13 给出 α' 的值到 40 位数字。

13. 在计算了相当麻烦的求和

$$\sum_{k=1}^{n}k^2H_{n+k}=\frac{n(n+1)(2n+1)}{6}(2H_{2n}-H_n)-\frac{n(n+1)(10n-1)}{36}$$

之后，我们得到答案

$$\widetilde{C}_N = \frac{4}{3}N - \frac{2}{3}(2N+1)(H_{2n} - H_n) + \frac{5}{6} - \frac{1}{3}(N+1)^{-1} \approx .409N$$

14. 我们可以假定 $x_1 \leqslant x_2 \leqslant \cdots \leqslant x_n$；则当 $y_{a_1} \leqslant y_{a_2} \leqslant \cdots \leqslant y_{a_n}$ 时，出现极大值，而当 $y_{a_1} \geqslant \cdots \geqslant y_{a_n}$ 时出现极小值，这是根据类似于定理 S 中所作的论证得到的。

15. 如同在定理 S 中那样论证，排列 $R_1R_2\cdots R_N$ 是最优的充要条件是

$$P_1/L_1(1-P_1) \geqslant \cdots \geqslant P_N/L_N(1-P_N)$$

16. 当且仅当 $T_1/(1-p_1) \leqslant \cdots \leqslant T_N/(1-p_N)$ 时预期的时间 $T_1 + p_1T_2 + p_1p_2T_3 + \cdots + p_1p_2\cdots p_{N-1}T_N$ 被极小化。[*BIT* **3** (1963), 255～256；James R. Slagle 得到了一些有趣的推广，*JACM* **11** (1964), 253～264。]

17. 以递增的截止时间（它们同分别的时间 T_i 无关）的顺序做诸工作！[当然，实际上某些作业比其它作业更重要，因此我们可能要使极大的加权延缓极小化。或者，我们可能希望使和数 $\sum_{i=1}^{n}\max(T_{a_1} + \cdots + T_{a_i} - D_{a_i}, 0)$ 极小化。看来这些问题中没有一个是有一个简单的解的。]

18. 设 $h = [s \text{ 存在}]$。$A = \{j \mid q_j < r_j\}$，$B = \{j \mid q_j = r_j\}$，$C = \{j \mid q_j > r_j\}$，$D = \{j \mid t_j > 0\}$；则对于 (q, r) 要排的和 $\sum_{i,j}p_ip_jd_{|i-j|}$ 减去对于 (q', r') 要排的对应的和等于

$$2\sum_{i\in A, j\in C}(q_i - r_i)(q_j - r_j)(d_{|i-j|} - d_{h+1+2k-i-j}) +$$
$$2\sum_{i\in C, j\in D}(q_i - r_i)t_j(d_{h+2k-i+j} - d_{i-1+j})$$

这是正的，除非 $C = \varnothing$ 或 $A \cup D = \varnothing$。由于当 $m = 0, 1$ 时它们左-右双重性，风琴管的那些安排法是仅有的、不可能被这个构造改进的排列，由此即得出所希望的结果。

[这个结果实质上是 G. H. Hardy、J. E. Littlewood 以及 G. Pólya 给出的*Proc. London Math. Soc.* (**2**), **25** (1926)265～282。他们证明，事实上，在诸 p 和诸 q 所有独立的安排，当诸 p 和诸 q 都在一致的风琴管次序下时，$\sum_{i,j}p_iq_jd_{|i-j|}$ 达到极小值。关于进一步的评述和推广，见他们的专著*Inequalities* (剑桥大学出版社，1934)，第 10 章。]

19. 所有安排都是同样好的。假定 $d(j, j) = 0$，我们有

$$\sum_{i,j}p_ip_jd(i,j) = \frac{1}{2}\sum_{i,j}p_ip_j(d(i,j) + d(j,i)) = \frac{1}{2}c$$

[对于 $i \neq j$ 的特殊情况 $d(i,j) = 1 + (j-i) \bmod N$ 是 K. E. Iverson 在*A Programming Language* (New York: Wiley, 1962), 138 中给出的。R. L. Baber (*JACM* **10**

(1963),478～486,已经研究了一条磁带可以向前读,重绕或不读而倒退 k 个块时,同磁带的查找有关的某些其它问题。W. D. Frazer 发现,如果允许我们复制文件中的某些信息,则有可能相当大量地减少查找时间;关于一个类似问题的经验解法参考 E. B. Eichelberge, W. C. Rodgers 和 E. W. Stacy, *IBM J. Research & Development* **12** (1968),130～159。]

20. 如同习题 18 中那样,以 $m=0$ 或 $m=h=1$ 从 (q,r) 进行到 (q',r') 给出了

$$\sum_{i\in A,j\in C}(q_i-r_i)(q_j-r_j)(d_{|i-j|})-\min(d_{h+1+2k-i-j},d_{i+j-1}))$$

的纯变化,除 A 或 C 为 $\varnothing$ 外它都是正的。由循环对称性得出,仅有的最优安排是风琴管配置的循环移位[对于具有相同答案的一类不同的问题,见 T. S. Motzkin 和 E. G. Straus, *Proc. Amer. Soc.* **7** (1956)1014～1021]。

21. 这个问题实际上是由 L. H. Harper 首先解决的, *SIAM* **12** (1964),131～135,关于推广及其它工作的参考文献,见 *J. Applied Probability* **4** (1967),397～401。

22. 大小为 1000 的一个优先队列(比如说,表示为一个堆,见 5.2.3 小节)。在这个队中插入前 1000 个记录,并把具有最大 $d(K_i,K)$ 的元素放在前端。对于满足 $d(K_i,K)<d$(队的前端,K)的每一个后继 K_i 以 R_i 代替前端元素,并对队进行重新调整。

6.2.1 小节

1. 归纳地证明,每当我们达到步骤 B2 时 $K_{l-1}<K<K_{u+1}$;而且每当达到 B3 时,$l\leqslant i\leqslant u$。

2. (a,c)否;如果 $l=u-1$ 和 $K>K_u$ 则它循环。(b)是,它是行的!。但当不存在 K 时,将经常有对于 $l=u$ 和 $K<K_u$ 的循环。

3. 这是对于 $N=3$ 的算法 6.1T。在一次成功的查找中,该算法平均作 $(N+1)/2$ 次比较;在一次不成功的查找中,它作 $N/2+1-1/(N+1)$ 次。

4. 对于 $N=127$ 它必然是一次不成功的查找;因此由定理 B,答案是 $138u$。

5. 程序 6.1Q′有 $1.75N+8.5-(N \bmod 2)/4N$ 的平均运行时间;这胜过程序 B 当且仅当 $N\leqslant 44$。[仅仅对于 $N\leqslant 11$ 它胜过程序 C。]

7. (a)肯定不是。(b)算法 U 中磁带圆括弧的注释将始终成立,所以它将是有效的,但当 N 为奇数时仅当 $K_0=-\infty$ 和 $K_{N+1}=+\infty$ 两者都出现时才是如此。

8. (a)N。通过归纳法证明这一点是有趣的,注意,如果我们以 $N+1$ 来代替 N,则诸 δ 中恰有一个增值。[关于一个推广,见 *AMM* **77** (1970),884。](b) 极大值 $=\sum_j\delta_j=N$;极小值 $=2\delta_1-\sum_j\delta_j=N \bmod 2$。

9. 当且仅当 $N=2^k-1$。

10. 使用"宏扩展"程序，其中包括诸 DELTA；因此，对于 $N=10$：

```
START   ENT1   5
        LDA    K
        CMPA   KEY,1
        JL     C3A
C4A     JE     SUCCESS          C3A   EQU    *
        INC1   3                      DEC1   3
        CMPA   KEY,1                  CMPA   KEY,1
        JL     C3B                    JGE    C4B
C4B     JE     SUCCESS          C3B   EQU    *
        INC1   1                      DEC1   1
        CMPA   KEY,1                  CMPA   KEY,1
        JL     C3C                    JGE    C4C
C4C     JE     SUCCESS          C3C   EQU    *
        INC1   1                      DEC1   1
        CMPA   KEY,1                  CMPA   KEY,1
        JE     SUCCESS                JE     SUCCESS
        JMP    FAILURE                JMP    FAILURE
```

[习题 23 表明大多数"JE"指令都可消去，同时产生大约长 $6\lg N$ 行的程序，它只花费大约 $4\lg N$ 个时间单位；但仅仅对于 $N\geqslant1000$（近似地）这个程序才是更快的。]

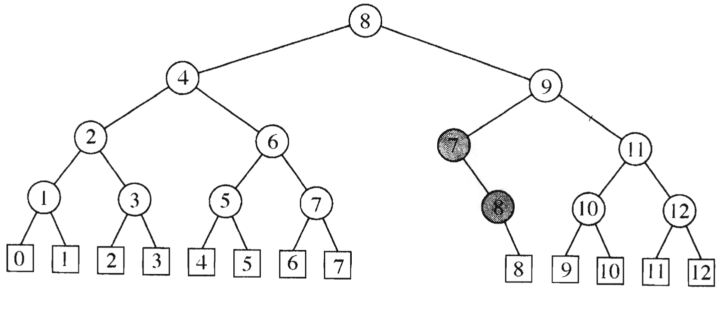
11. 考虑相应的树，例如图 6：当 N 是奇数时，这个根的左子树是右子树的一个镜面映像，所以 $K<K_i$ 出现的次数同 $K>K_i$ 一样多；平均有，$C1=\frac{1}{2}(C+S)$，而且 $C2=\frac{1}{2}(C-S)$，$A=\frac{1}{2}(1-S)$。当 N 是偶数时，这株树和对于 $N+1$ 的一株树相同，其中所有标号减 1，但⓪变成冗余；平均说来，命 $k=\lfloor\lg N\rfloor$。我们有

$$C1=\frac{C+1}{2}-\frac{k}{2N},\qquad C2=\frac{C-1}{2}+\frac{k}{2N},\qquad A=0,\qquad \text{如果 } S=1$$

$$C1=\frac{(k+1)N}{2(N+1)},\qquad C2=\frac{(k+1)(N+2)}{2(N+1)},\qquad A=\frac{N}{2(N+1)},\qquad \text{如果 } S=0$$

（正文中指出了 C 的平均值。）

12.



13.

$N=$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$C_N=$	1	$1\frac{1}{2}$	$1\frac{1}{3}$	$2\frac{1}{4}$	$2\frac{1}{5}$	$2\frac{2}{6}$	$2\frac{3}{7}$	$3\frac{1}{8}$	3	3	3	$3\frac{2}{12}$	$3\frac{3}{13}$	$3\frac{3}{14}$	$3\frac{4}{15}$	$4\frac{1}{16}$
$C'_N=$	1	$1\frac{2}{3}$	2	$2\frac{3}{5}$	$2\frac{4}{6}$	3	3	$3\frac{6}{9}$	$3\frac{6}{10}$	$3\frac{8}{11}$	$3\frac{8}{12}$	4	4	4	4	$4\frac{13}{17}$

14.一个思想是求使得 $N+M$ 有形式 $F_{k+1}-1$ 的最小的 $M\geqslant 0$,然后在步骤 F1 中以 $i\leftarrow F_k-M$ 开始,而且在步骤 F2 的开头处插入"如果 $i\leqslant 0$,则转向 F4。"一个更好的解决方法是,对于斐波那契的情况采取 Shar 的思想:如果最初比较的结果是 $K>K_{F_k}$,则置 $i\leftarrow i-M$ 并转到步骤 F4(从这以后正常进行)。这就避免了内循环中的额外时间。

15.外部节点出现于级$\lfloor k/2\rfloor$到 $k-1$ 上;这些级之间的差别大于 1,除非当 $k=0,1,2,3,4$ 时。

16.在 2.3.2 节的"自然对应"之下,如果我们撤销线性图表最顶部的节点,则阶为 k 且左右颠倒的斐波那契树,是对应于直到第 k 个月止的线性图表的二叉树。

17.设路径长度是 $k-A(n)$;则当 $0<m<F_{j-1}$时,$A(F_j)=j$ 和 $A(F_j+m)=1+A(m)$。

18.成功的查找:$A_k=0$, $C_k=(3kF_{k+1}+(k-4)F_k)/5(F_{k+1}-1)-1$, $C1_k=C_{k-1}(F_k-1)/(F_{k+1}-1)$。不成功的查找:$A'_k=F_k/F_{k+1}$, $C'_k=(3kF_{k+1}+(k-4)F_k)/5F_{k+1}$, $C1'_k=C'_{k-1}F_k/F_{k+1}+F_{k-1}/F_{k+1}$。$C_2=C-C_1$。(关于有关的递推式的解,参见习题1.2.8-12。)

20.(a) $b=p^{-p}q^{-q}$。(b) 至少有两个错误。第一个大错误是该因子不是一线性函数,所以它不能简单地"平均之"。实际上剩下 pN 个元素的概率为 p,剩下 qN 个的概率为 q,所以我们可预期剩下$(p^2+q^2)N$ 个;于是,平均的减少因子实际上是 $1/(p^2+q^2)$。现在在 k 次迭代之后的减少因子是 $1/(p^2+q^2)^k$,但我们不能作出结论说 $b=1/(p^2+q^2)$,因为找出某些项所需要的迭代次数要比找出其它的项多得多。这是第二个失误。[作出似是而非的概率论断是非常容易的,但我们必须时刻防止这样的圈套!]

21.它是不可能的,因为这个方法依赖于键码的值。

22. *FOCS* **17**(1976),173~177。也请参见 Y. Perl, A. Itai 和 H. Avni, *CACM* **21**(1978),550~554; G. H. Gonnet, L. D. Rogers 和 J. A. George *Acta. Informatica* **13**(1980),39~52;G. Louchard,*RAIRO Inform. Théor* **17**(1983),365~385;*Computing* **46**(1991),193~222。方差是 $O(\log\log N)$。G. Marsaglia 和 B. Narasimhan 所作的广泛的经验测试,*Computer and Math.* **26**,**8**(1993),31~42 表明,平均的表访问次数非常接近于 $\lg\lg N$,如果查找不成功加上大约 0.7。例如当 $N=2^{20}$时,在一个随机表格中的一次随机成功查找花费大约 4.29 次访问,而一次随机不成功查找花费大约 5.05。

23.当大于等于时转向右边,当小于时转向左边;当达到节点$\boxed{i}$时,由(1)得出

$K_i \leqslant K < K_{i+1}$，所以关于相等性的最后一次测试将区别成功或失误。（键码 $K_0 = -\infty$将总是存在的。）

算法 C 将被改变成，如果在步骤 C2 中 $K = K_i$，则转到 C4。在 C3 中如果 DELTA$[j]=0$，则置 $i \leftarrow i-1$ 并转到 C5。在 C4 中如果 DELTA$[j]=0$，则直接地转到 C5。增加一个新步骤 C5："如果 $K = K_i$，则这算法成功地结束，否则它以失败告终。"[除非 $N > 2^{26}$，否则这不会加快程序 C，平均成功的查找时间从 $(8.5\ \lg N - 6)u$ 变成 $(8\lg N + 7)u$。]

24. 可以把诸键码安排成使得我们首先置 $i \leftarrow 1$，然后按 $K < K_i$ 或 $K > K_i$ 置 $i \leftarrow 2i$ 或 $2i+1$；当 $i > N$ 时，这个查找是不成功的。例如当 $N = 12$ 时，键码的安排必须是

$$K_8 < K_4 < K_9 < K_2 < K_{10} < K_5 < K_{11} < K_1 < K_{12} < K_6 < K_3 < K_7$$

对于在 MIX 上编的程序，这个方法大约将花费 $6\ \lg N$ 个时间单位，所以它比程序 C 更快。惟一的缺点是首先建立表格时要有点技巧。

25. (a) 由于 $a_0 = 1 - b_0$，$a_1 = 2a_0 - b_1$，$a_2 = 2a_1 - b_2$，等等，我们有 $A(z) + B(z) = 1 + 2zA(z)$。通过考虑 $A(1)$，$B(1)$，$B(\frac{1}{2})$，$A'(1)$和 $B'(1)$，从这个关系立即得出在2.3.4.5节中导出的若干公式。如果我们使用两个变量来区别一条通路的向左走和向右走，则我们得到更一般的结果 $A(x,y) + B(x,y) = 1 + (x+y)A(x,y)$，这是在 t 叉树中成立的一个公式的特殊情况[参见 R. M. Karp, *IRE Transactions*, **IT-7**(1961), 27～38]。

(b) $\mathrm{var}(g) = ((N+1)/N)\mathrm{var}(h) - ((N+1)/N^2)\ \mathrm{mean}\ (h)^2 + 2$。

26. 如果我们适当地排列左边和右边，则具有一个完全的 k 级分布的三条带多阶段合并的合并树就是 $k+1$ 阶的斐波那契树。（重画 5.4.4 节图 76 的多阶段树的图形，并颠倒 A 和 C 的左和右子树，即得图 8。）

27. 2^k 个结果中至多有 $k+1$ 个会出现，因为我们可以把下标排序，使得 $K_{i_1} < K_{i_2} < \cdots < K_{i_k}$。于是这个查找可以通过一株在每个节点处至多有$(k+1)$路分支的树来描述。在第 m 步中可以找到的项目数至多是$k(k+1)^{m-1}$；因此平均比较次数至少是 N^{-1}乘以多重集合$\{k\cdot 1, k(k+1)\cdot 2, k(k+1)^2\cdot 3, \cdots\}$的最小的 N 个元素之和。当 $N \geqslant (k+1)^n - 1$ 时，平均比较次数$\geqslant ((k+1)^n - 1)^{-1}\sum_{m=1}^{n} k(k+1)^{m-1}m > n - 1/k$。

28. [*Skrifter udgivne af Videnskabs-Selskabet i Christiania*, Mathematisk-Naturviden-skabelig Klasse(1910), No. 8；再版于 Thue 的*Selected Mathematical Papers* (Oslo: Universitetsforlaget, 1977), 273～310。] (a) T_n 有 $F_{n+1} + F_{n-1} = F_{2n}/F_n$ 个叶(这是所谓的 Lucas 数 $L_n = \phi^n + \hat{\phi}^n$)。(b) 这个公理说 $T_0(T_2(x)) = T_1(x)$，而且当 $m = 1$ 或 $n = 1$ 时，我们显然有 $T_m(T_n(x)) = T_{m+n-1}(x)$。通过对 n 用归纳法，当 $m = 0$ 时这个结果成立；例如，$T_0(T_3(x)) = T_0(T_2(x) * T_1(x)) =$

$T_0(T_1(T_2(x)) * T_0(T_2(x))) = T_0(T_2(T_2(x))) = T_2(x)$。最后,我们可以使用对 m 的归纳法。

29. 假定 $K_0 = -\infty$ 和 $K_{N+1} = K_{N+2} = \infty$。首先对于 $K_2 < K_4 < \cdots$ 进行一次二叉查找;这至多花费$\lfloor \lg N \rfloor$次比较。如果不成功,它确定满足 $K_{2j-2} < K < K_{2j}$ 的一个区间。如果 $2j = N + 2$ 则 K 不存在。否则,对于 K_{2j-1}的一个二叉查找将确定使得 $K_{2i-2} < K_{2j-1} < K_{2i}$ 的 i。于是,或者 $K = K_{2i-1}$或者 K 不存在[参见*Theor. Comp. Sci.* **58** (1988),67]。

30. 设 $n = \lfloor N/4 \rfloor$。由 $K_1 < K_2 < \cdots < K_N$ 开始,通过把 $K_1, K_3, \cdots, K_{2n-1}$和 $K_{2n+1}, K_{2n+3}, \cdots, K_{4n-1}$的一个排列进行交换,我们可以把 $K_1, K_3, \cdots, K_{2n-1}$放入任何想要的次序。这个安排满足前面习题的条件。现在我们设 $K_1 < K_3 < \cdots < K_{2^{t+1}-3}$表示所有可能的 t 个二进位的数之间的边界,而且按照 $x_1, x_2, \cdots, x_m$ 的值我们把$K_{2^{t+1}-1}, K_{2^{t+1}+1}, \cdots, K_{2^{t+1}+2m-3}$插入到这些"围栅位置"之间。例如,如果 $m = 4, t = 3, x_1 = (001)_2, x_2 = (111)_2$ 以及 $x_3 = x_4 = (100)_2$,则所求次序是

$$K_1 < K_{15} < K_3 < K_5 < K_7 < K_{19} < K_{21} < K_9 < K_{11} < K_{13} < K_{17}$$

(我们也可以令 K_{21} 在 K_{19}之前。)在子数组 $K_1 < K_3 < \cdots < K_{2^{t+1}-3}$中对于 $K_{2^{t+1}+2j-3}$的一次二叉查找现在将从左到右地找 x_j 的二进位。[参见 Fiat,Munro,Naor,Schäffer,Schmidt 及 Siegel, *J. Comp. Syst. Sci* **43** (1991),406~424。]

6.2.2 小节

1. 使用一个表头节点,并设 ROOT = RLINK(HEAD);以 P←HEAD 在步骤 T4 处开始这个算法。步骤 T5 就如同当 $K >$ KEY(HEAD)时那样执行。[于是改变程序 T 的行 04 和 05 为"ENT1 ROOT;CMPA K"]。

2. 在步骤 T5 中,置 RTAG(Q)←1。还有,当插入左边时,置 RLINK(Q)←P;当插入右边时,置 RLINK(Q)←RLINK(P)及 RTAG(P)←0。在步骤 T4 中,把测试"RLINK(P)≠Λ"改变成为"RTAG(P)≠0"。[如果诸节点被成功地插入到递增的诸位置 Q,而且如果所有的删去都是后进先出,则可删去 RTAG 字段,因为当且仅当 RLINK(P)<P 时,RTAG(P)为 1。类似的注释对于同时的左和右穿线亦可应用。]

3. 我们可以以一个正确的地址代替 Λ,并在这个算法的开始处置 KEY(Λ)←K;然后可从内循环撤销对于 LLINK 或 RLINK = Λ 的判断。然而为了进行一个正确的插入,我们需要引进跟随 P 的另一个指针变量;这可以在不失去所述速度的优点之下进行,并通过如程序 6.2.1F 中那样重复代码的方法来完成。于是 MIX 的时间将被减少成大约 $5.5C$ 单位。

4. 对于 $N \geq 2^n - 1$, $C_N = 1 + (0 \cdot 1 + 1 \cdot 2 + \cdots + (n-1)2^{n-1} + C'_{2^n-1} + \cdots + C'_{N-1}/N = (1 + 1/N)C'_N - 1$。对于 $N \geq 2^n - 1$,这些方程的解是 $C'_N = 2(H_{N+1} - H_{2^n}) + n$,它节省了 $2H_{2^n} - n - 2 \approx n(\ln 4 - 1)$次比较。对于 $n = 1,2,3,4$,实际的改

进分别是 $0,\frac{1}{6},\frac{61}{140},\frac{274399}{360360}$；于是对于小的固定的 n 获利是相当小的。[关于和一个等价的排序问题的更详细的推导，见 Frazer 和 McKellar, *JACM* **17** (1970), 502。]

5. (a) 第一个元素必须是 CAPRICORN；然后我们把产生左子树的方式数乘以产生右子树的方式数再乘以 $\binom{10}{3}$，即为把这两个序列搅混到一起的方式数。于是答案成为

$$\binom{10}{3}\binom{2}{0}\binom{1}{0}\binom{0}{0}\binom{6}{3}\binom{2}{0}\binom{1}{0}\binom{0}{0}\binom{2}{1}\binom{0}{0}\binom{0}{0} = 4800$$

[一般地说，这个答案是 $\binom{l+r}{r}$ 对于所有节点的乘积，其中 l 和 r 代表这个节点的左和右子树的大小。这等于 $N!$ 除以子树大小的乘积。它和习题 5.1.4-20 中的公式一样；其实，如果我们以 k 代替查找树中的 a_k(用习题 6 的记号)，则在产生一株具体的查找树的诸排列和在该习题中统计的诸"拓扑"排列之间，有明显的一一对应。] (b) $2^{N-1}=1024$；在除开最后一步之外的每一个步骤中，插入最小的或最大的剩余键码。

6. (a) 对于其费用是 k 的 P_{nk} 个排列 $a_1\cdots a_{n-1}a_n$ 中的每一个，按照 $a_j<m$ 或 $a_j\geqslant m$ 构造 $n+1$ 个排列 $a'_1\cdots a'_{n-1}ma'_n$，其中 $a'_j=a_j$；或 a_j+1。[参见 1.2.5 节，方法 2。] 如果 $m=a_n$ 或 a_n+1，则这个排列费用为 $k+1$，否则它的费用是 k。(b) $G_n(z)=(2z+n-2)(2z+n-3)\cdots(2z)$。因此

$$P_{nk} = \left[{n-1 \atop k}\right]2^k$$

实质上，这个生成函数是由 W. C. Lynch 得到的[*Comp. J.* (1965), 299～302]。(c) 诸概率的生成函数是 $g_n(z)=G_n(z)/n!$。这是诸简单概率生成函数的一个乘积，所以 C'_{n-1} 的方差是

$$\mathrm{var}(g_n) = \sum_{k=0}^{n-2}\mathrm{var}\left(\frac{2z+k}{2+k}\right) = \sum_{k=0}^{n-2}\left(\frac{2}{k+2}-\frac{4}{(k+2)^2}\right) = 2H_n - 4H_n^{(2)} + 2$$

[由习题 6.2.1-25(b)，我们可以利用 C'_n 的均值和方差计算 C_n 的方差，它是 $(2+10/n)H_n-4(1+1/n)(H_n^{(2)}+H_n^2/n)+4$；这个公式由 G. D. Knott 给出。]

7. 将进行同第 k 个最大元素的比较当且仅当该元素出现在第 m 个元素之前和介于第 k 个和第 m 个之间的所有元素之前；这以 $1/(|m-k|+1)$ 的概率出现。对 k 求和给出答案 $H_m+H_{n+1-m}-1$。[*CACM* **12** (1969), 77～80；也见 L. Cuibas, *Acta Informatica* **4** (1975), 293～298。]

8. (a) $g_n(z)=z^{n-1}\sum_{k=1}^{n}g_{k-1}(z)g_{n-k}(z)/n$, $g_0(z)=1$。

(b) $7n^2-4(n+1)^2H_n^{(2)}-2(n+1)H_n+13n$。[P. F. Windley, *Comp. J.* **3** (1960), 86, 给出了一些递推关系，由这些递推关系，可以从数值上计算这个方差，但他未得到这个解。注意这个结果同习题 6 的答案中所述的 C_n 的方差没有简单的关

联。]

10. 例如，键码的每个字 x 都可为 $ax \bmod m$ 所代替，其中 m 是计算机字的大小而 a 是同 m 互素的一个随机乘数。可建议采用接近于 $(\phi-1)m$ 的一个值见(6.4 节)。树方法的灵活存储分配，使它较之其它散列代码方案更有吸引力。

11. $N-2$；但是这仅仅是在删去① $N(N-1)\cdots 2$ 时，以 $1/(NN!)$ 的概率出现。

12. 定理 H 的证明中 $\frac{1}{2}(n+1)(n+2)$ 个删去属于情况 1，所以答案是 $(N+1)/2N$。

13. 是。事实上定理 H 的证明表明，如果我们对于任何固定的 k，删去第 k 个插入的元素，则结果是随机的。(G. D. Knott[斯坦福大学博士论文，1975]已经证明，对于任何固定的序列 $k_1,\cdots,k_d$，在任意的随机插入序列之后紧接着逐次删去已插入的第 $(k_1,\cdots,k_d)$ 个元素，则结果是随机的。

14. 命 NODE(T)在级 k 上，且设 LLINK(T) = Λ，RLINK(T) = R_1，LLINK(R_1) = R_2，⋯，LLINK(R_d) = Λ，其中 $R_d \neq \Lambda$ 且 $d \geqslant 1$。设对于 $1 \leqslant i \leqslant d$，NODE($R_i$)在它的右子树中有 n_i 个节点。通过步骤 D $1\frac{1}{2}$，内部路径长度减少 $k+d+n_1+\cdots+n_d$；如果没有这个步骤，则它减少 $k+d+n_d$。

15. 11，13，25，11，12。[如果 a_j 是 $\{a_1,a_2,a_3\}$ 的(最小的，中间的，最大的)，则在删去之后得到树⟍(4,2,3)×4 次。]

16. 是；甚至像在定理 H 的证明中所定义的那样，对于排列的删去操作也是可交换的(如果我们忽略重新编号这一点)。如果在 X 和 Y 之间有一个元素，则删去显然是可交换的，因为这个操作仅受 X 和 Y 的相对位置，及它们的后继者位置的影响；而且在 X 的删去和 Y 的删去之间没有交互作用。另一方面，如果 Y 是 X 的后继，而且 Y 是最大的元素，则删去的两种次序都有简单地撤销 X 和 Y 的作用。如果 Y 是 X 的后继者以及 Z 是 Y 的后继者，则两种删去的次序都有以 Z 代替 X，Y 或 Z 的第一次出现的作用，而且删去这些元素在这个排列内的第二次和第三次出现。

18. 使用习题 1.2.7-14。

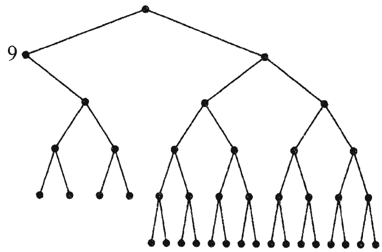
19. $2H_N-1-2\sum_{k=1}^{N}(k-1)^\theta/kN^\theta = 2H_N-1-2/\theta+O(N^{-\theta})$。[Pareto 分布 6.1-(13)也给出相同的渐近结果，直到 $O(n^{-\theta}\log n)$ 的范围内。]

20. 确实是。假定 $K_1<\cdots<K_N$，使得由算法 T 构造的树是退化的；如果比如说 $p_k=(1+((N+1)/2-k)\epsilon)/N$，则平均的比较次数是 $(N+1)/2-(N^2-1)\epsilon/12$，而最优树只需少于 $\lceil\lg N\rceil$ 次比较。

21. $\frac{1}{8},\frac{3}{20},\frac{9}{20},\frac{3}{20},\frac{1}{8}$。(大部分角是 30°，60°或 90°。)

22. 当 $d=2$ 时这是显然的，而对 $d>2$ 我们有 $r[i,j-1]\leqslant r[i+1,j-1]\leqslant r[i+1,j]$。

23.



[增加第一个节点的权将使它最终移到根的位置;这提示,动态地维持一株完全最优的树是困难的。]

24.设 c 是通过删去一株最优树的第 n 个节点得到的一株树的费用,则 $c(0,n-1)\leqslant c\leqslant c(0,n)-q_{n-1}$,这是因为删去操作总是把 $\boxed{n-1}$ 上移一级。还有 $c(0,n)\leqslant c(0,n-1)+q_{n-1}$,因为所述的替代产生最后一个费用的一株树。由此得出 $c(0,n-1)=c=c(0,n)-q_{n-1}$。

25.(a) 假设 $A\leqslant B$ 和 $B\leqslant C$,且 $a\in A,b\in B,c\in C,c<a$。如果 $c\leqslant b$,则 $c\in B$;因此 $c\in A$ 和 $a\in B$;因此 $a\in C$。如果 $c>b$,则 $a\in B$;因此 $a\in C$ 和 $c\in B$;因此 $c\in A$。(b)不难证明。

26.对于某个实数 $y\geqslant 0$ 和整数 $l>0$,每株树的费用有 $y+lx$ 的形式。有限个这样的函数的(对于所有树来取)极小值总有所述的形式。

27.(a) 习题 24 的答案(特别是 $c=c(0,n-1)$ 的事实)意味着 $R(0,n-1)=R(0,n)\setminus\{n\}$。(b) 如果 $l=l'$,则提示中的结果是显然的,否则设对于 $\boxed{n}$ 的各通路是

$$ⓡ_0, ⓡ_1, \cdots, \boxed{r_l} \quad 和 \quad ⓢ_0, ⓢ_1, \cdots, \boxed{s_{l'}}$$

由于 $r=r_0>s_0=s$ 和 $r_{l'}<s_{l'}=n$,我们可以求出一个级 $k\geqslant 0$ 使得 $r_k>s_k$ 和 $r_{k+1}\leqslant s_{k+1}$。由归纳法我们有 $r_{k+1}\in R(r_k,n)$,$s_{k+1}\in R(s_k,n)$,以及 $R(s_k,n)\leqslant R(r_k,n)$,因此 $r_{k+1}\in R(s_k,n)$和 $s_{k+1}\in R(r_k,n)$;便得到提示中的结果。

现在证明 $R'_h\leqslant R_h$,设 $r\in R'_h$,$s\in R_h$,$s<r$,而且考虑当 $x=x_h$ 时出现的诸最优树;我们必须有 $l\geqslant l_h$ 且可以假定 $l'=l_h$。为了证明 $R_h\leqslant R'_{h+1}$,设 $r\in R_h$,$s\in R'_{h+1}$,$s<r$,而且考虑当 $x=x_{h+1}$时出现的最优树;我们必然有 $l'\leqslant l_h$ 因而可以假定 $l=l_h$。

29.它是一株 YOU 在顶上,THE 在底下的退化树,平均需要 19.158 次比较。

Douglas A. Hamilton 已经证明,总有一株退化树是最坏的,因此存在求悲观的二分查找树的一个 $O(n^2)$算法。

30.参见 R. L. Wessner, *Information Processing Letters* **4** (1976),90~94;姚期智,*SIAM J. Algebraic and Discrete Methods* **3** (1982),532~540。

31.参见 *Acta Informatica* **1**(1972),307~310。

32. 当 M 足够大时，最优树必有所述形式且极小费用必是 M 乘以极小外部通路长度加上对于所述问题的解。

注：在答案 30 中所引用的 Wessner 的论文说明如何来求高度 $\leqslant L$ 的最优二分查找树。在 $p_1=\cdots=p_n=0$ 的特殊情况下，所述的结果是由胡德强和陈国财给出的，MRC 报告 1111（威斯康辛大学，1970）。A. M. Garsia 和 M. L. Wachs 证明，在这种情况下，如果 $\min_{k=1}^{n}(q_{k-1}+q_k)\geqslant \max_{k=0}^{n} q_k$，则所有外节点都将出现在至多两级，而且他们给出了一个算法，该算法只需 $O(n)$ 步即可求出一棵最优的二级树。

33. 对于所述的问题，请见 A. Itai, *SICOMP* **5** (1976), 9～18。关于其它可能性，参见 D. Spuler, *Acta Informatica* **31** (1994), 729～740。

34. 由 Stivling 的近似公式，如果 $p_1\cdots p_n\neq 0$，它等于 $2^{H(p_1,\cdots,p_n)N}(2\pi N)^{(1-n)/2}(p_1\cdots p_n)^{-1/2}(1+O(1/N))$。

35. 当 $2x=(1-p)/p$ 时右边的极小值出现，而且它等于 $1-p+H(p,1-p)$。但对于 $k=2$，由(20)，$H(p,q,r)\leqslant 1-p+H(p,1-p)$。

36. 首先我们证明提示，这个提示是由 Jensen 给出的[*Acta Math.* **30**(1906), 175～193]。如果 f 是凹的，则函数 $g(p)=f(px+(1-p)y)-pf(x)-(1-p)f(y)$ 是凹的并满足 $g(0)=g(1)=0$。如果 $g(p)<0$ 和 $0<p<1$，则由中值定理必有一个值 $p_0<p$ 且 $g'(p_0)<0$ 和一个值 $p_1>p$ 且 $g'(p_1)>0$；但这和凹性相矛盾。因此对于 $0\leqslant p\leqslant 1$，$f(px+(1-p)y)\geqslant pf(x)+(1-p)f(y)$。这是在几何上显然的一个事实。现在我们通过归纳法证明 $f(p_1x_1+\cdots+p_nx_n)\geqslant p_1f(x_1)+\cdots+p_nf(x_n)$，因为若 $n>2$ 则 $f(p_1x_1+\cdots+p_nx_n)\geqslant p_1f(x_1)+\cdots+p_{n-2}f(x_{n-2})+(p_{n-1}+p_n)f((p_{n-1}x_{n-1}+p_nx_n)/(p_{n-1}+p_n))$。

由引理 E，我们有

$$H(XY)=H(X)+\sum_{i=1}^{m}p_iH(r_{i1}/p_i,\cdots,r_{in}/p_i)$$

而后一个和是 $\sum_{j=1}^{n}\sum_{i=1}^{m}p_if(r_{ij}/p_i)\leqslant\sum_{j=1}^{n}f(\sum_{i=1}^{m}r_{ij})=H(y)$，其中 $f(x)=x\lg(1/x)$ 是凹的。

37. 由习题 3.3.2-26 的部分(a)，我们有 $\Pr(P_1\geqslant s)=(1-s)^{n-1}$。因此

$$\mathrm{E}H(P_1,\cdots,P_n)=n\mathrm{E}P_1\lg(1/P_1)=n\int_0^1(1-s)^{n-1}\mathrm{d}(s\lg(1/s))=-(A+B)/\ln 2,$$

其中 $A=n\int_0^1(1-s)^{n-1}\mathrm{d}s=1$ 而且由习题 1.2.7-13，

$$B=n\int_0^1(1-s)^{n-1}\ln s\,\mathrm{d}s=\sum_{k=1}^{n}\binom{n}{k}(-1)^ks^k\left(\frac{1}{k}-\ln s\right)\Bigg|_0^1=-H_n$$

因此答案是 $(H_n-1)/\ln 2$。（这是 $\lg n+(\gamma-1)/\ln 2+O(n^{-1})$，非常接近于极大的熵 $H(\frac{1}{n},\cdots,\frac{1}{n})=\lg n$。因此 $H(p_1,\cdots,p_n)$ 以很高的概率是 $\Omega(\log n)$。）

38. 如果 $s_{k-1}=s_k$,我们有 $q_{k-1}=p_k=q_k=0$; 见(26)。构造对于 $n-1$ 个概率 $(p_1,\cdots,p_{k-1},p_{k+1},\cdots,p_n;q_0,\cdots,q_{k-1},q_{k+1},\cdots,q_n)$的一株树,并以一个 2 叶子树代替叶$\boxed{k-1}$。

39. 我们可以像在定理 M 中那样论证,如果 $0<w_1\leqslant w_2\leqslant\cdots\leqslant w_n$ 而且 $s_k=w_1+\cdots+w_k$,因为 $w_k\geqslant 2^{-t}$意味着当诸权是有序的时,$s_{k-1}+2^{-t}\leqslant s_k\leqslant s_{k+1}-2^{-t}$;因此我们有$|\sigma_k|<1+\lg(1/w_k)$。[这个结果,连同匹配的下限 $H(w_1,\cdots,w_k)$]一起,是 1948 年 Shannon 的开创性论文的定理 9。]

40. 如果 $k=s+3$,则所述的重新安排把费用由 $q_{k-1}l+q_kl+q_{k-2}l_{k-2}$改变成为 $q_{k-2}l+q_{k-1}l+q_kl_{k-2}$,所以纯变化是$(q_{k-2}-q_k)(l-l_{k-2})$;如果 $l<l_{k-2}$,则这为负,因为 $q_{k-2}>q_k$。

类似地,如果 $k\geqslant s+4$,则这重新安排使费用改变

$$\delta = q_{s+1}(l-l_{s+1})+q_{s+2}(l-l_{s+2})+q_{s+3}(l_{s+1}-l_{s+3})+\cdots+q_{k-2}(l_{k-4}-l_{k-2})+ q_{k-1}(l_{k-3}-l)+q_k(l_{k-2}-l)$$

我们有 $q_{s+1}>q_{s+3},q_{s+2}>q_{s+4},\cdots,q_{k-2}>q_k$。因此我们求得

$$\delta\leqslant(q_{k-2}-q_k)(l-l_{k-2})+(q_{k-3}-q_{k-1})(l-l_{k-3})\leqslant 0;$$

例如,当 $k-s$ 为偶数时,我们有

$$\delta\leqslant q_{k-3}(l-l_{s+1})+q_{k-2}(l-l_{s+2})+q_{k-3}(l_{s+1}-l_{s+3})+\cdots+ q_{k-2}(l_{k-4}-l_{k-2})+q_{k-1}(l_{k-3}-l)+q_k(l_{k-2}-l)$$

而且当 $k-3$ 为奇数时,类似的推导也有效。由此得出,除非 $l_{k-2}=l$,否则 δ 为负。

41. E F G H T U X Y Z V W B C D A P Q R J K L M I N O S ␣。

42. 设 $q_j=\mathtt{WT}(\mathtt{P}_j)$。关键之点在于步骤 C2~C6 的动作使得所有 q 大于或等于 $q_{k-1}+q_k$ 的初始值。

43. 调用递归过程 mark($\mathtt{P}_1$,0),其中 mark(P,l)意味着下列操作:

LEVEL(P) $\leftarrow l$;
如果 LLINK(P) $\neq\Lambda$,则 mark(LLINK(P),$l+1$);
如果 RLINK(P) $\neq\Lambda$,则 mark(RLINK(P),$l+1$)。

44. 置全局变量 $t\leftarrow 0$, $m\leftarrow 2n$,并且调用递归子程序 build(1),其中 build(1)指的是下列:

置 $j\leftarrow m$;
如果 LEVEL($\mathtt{X}_t$) $=l$ 则置 LLINK($\mathtt{X}_j$) $\leftarrow t$ 和 $t\leftarrow t+1$,
否则置 $m\leftarrow m-1$,LLINK($\mathtt{X}_j$) $\leftarrow \mathtt{x}_m$ 和 build($l+1$);
如果 LEVEL($\mathtt{X}_t$) $=l$,则置 RLINK($\mathtt{X}_j$) $\leftarrow t$ 和 $t\leftarrow t+1$,
否则置 $m\leftarrow m-1$,RLINK($\mathtt{X}_j$) $\leftarrow \mathtt{X}_m$,和 build($l+1$)。

变量 j 是局部于 build 子程序的。[这漂亮的解是由 R. E. Tarjan 给出的,*SICOMP* **6** (1997),639。]告诫:如果数 $l_0,\cdots,l_n$ 不对应于任何二叉树,则这个算法

将永远循环。

45. 把工作数组 $p_0,\cdots,p_t$ 作为一个双重链接表，这个双重链表也有一个平衡树的链接(参见 6.2.3 节)。如果 2 递降的权是 $q_0,\cdots,q_t$ 且 q_j 是树的根，则基于 q_j 和 q_{j+1}的值，我们可以判定在这个树中是向左进行还是向右进行；双重链接提供了对于 q_{j+1}的即时访问。(不需要 RAND 字段；转动保持对称次序，因此它不要求对双重链接的任何改动。)

胡德强和 Morgenthaler 已经给出了可以在 $O(n)$的时间内解决问题的若干权的类。*Lecture Notes in Comp. Sci.* **1120** (1996)，234～243；一般地说还不知道$O(n)$步是否充分。

46. 参见*IEEE Trans.* **C-23** (1974)，268～271；也可参见习题 6.2.3-21。

47. 参见 Altenkamp 和 Mehlhorn，*JACM* **27** (1980)，412～427。

48. 不要让对于 $N=3$ 的情况[Jonassen 和 Knuth，*J. Comp. Syst. Sci.* **16** (1978)，301～322]或 N=4[Baeza-Yates，*BIT* **29** (1989)，378～390]的复杂分析使你恐惧，想开些！Louchard，Randrianarimanana 和 Schott，*Theor. Comp. Sci.* **93** (1992)，201～225 已经报告了某些进展。

49. 这个问题首先是由 J. M. Robson [*Australian Comp. J.* **11** (1979)，151～153]，B. Pittel [*J. Math. Anal. Applic* **103** (1984)，461～480] 以及 Luc Devroye [*JACM* **33** (1986)，489～498；*Acta. Inf.* **24** (1987)，277～298]考查的。他们得到了极限公式，当 $n\to\infty$时这些公式以→1 的概率成立。参见由 H. M. Mahmoud，*Evolution of Random Search Trees* (Wiley，1992)，第 2 章。Bruce Reedm [*STOC* **32** (2000)，479～483]证明了平均高度是 $\alpha\ln n-(3\alpha\ \ln\ln\ n)/(2\alpha-2)+O(1)$，而方差是 $O(1)$，其中

$$\alpha = 1/T(1/2e) \approx 4.31107040700100503504707609644689027839l6 -$$

而 $T(z)=\sum_{n=1}^{\infty}n^{n-1}z^n/n!$ 是树函数。

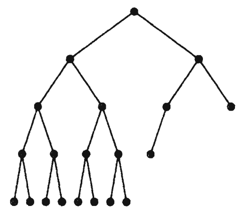
6.2.3 小节

1. 节点的对称次序必须通过变换加以保持，否则我们将失去一株二分查找树。

2. B(S)=0 仅当 S 指向树根时才能发生(在步骤 A3 或 A4 中它决不曾被改变)，而且从 S 到插入点的所有节点是平衡的。

3. 设 ρ_h 是在高度为h 的平衡树中不平衡节点的最大可能比率。于是 $\rho_1=0$，$\rho_2=\dfrac{1}{2}$，$\rho_3=\dfrac{1}{2}$。我们将证明 $\rho_h=(F_{h+1}-1)/(F_{h+2}-1)$。设 T_h 是使ρ_h 极大化的一株树；则可以假设它的左子树有高度 $h-1$ 且它的右子树有高度 $h-2$，因为如果两个子树的高度都是 $h-1$，则比率将小于 ρ_{h-1}。于是 T_h 的比率至多是$(\rho_{h-1}N_l+\rho_{h-2}N_r+1)/(N_l+N_r+1)$，其中在(左，右)子树中有$(N_l,N_r)$个节点，当$(N_l,N_r)$取它们的极小值时，这个公式取它的极大值；因此我们可以假定 T_h 是一株斐波

那契树。而且由习题 1.2.8-28，$\rho_h < \phi - 1$。



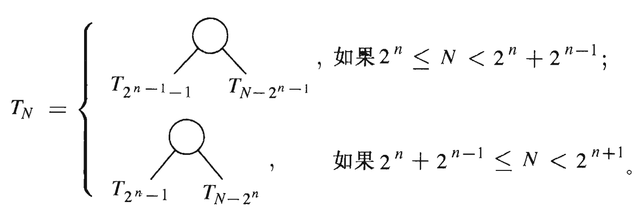
4. 当 $n=7$ 时有更大的路径长度。[注：C. C. Foster, *Proc. ACM Nat. Conf.* **20** (1965), 197～198，给出了构造具有极大路径长度的 N 节点平衡树的一个不正确的过程；Edward Logg 已经发现，Foster 的图 3 在 24 个步骤之后给出了一个非最优的结果(编号为 22 的点可被撤销以留下编号为 25 的节点。)]

然而当 a 是任何非负常数时，阶为 h 的斐波那契数确实在所有高度为 $h-1$ 的平衡树 T 中极小化 $(h+a)N-$(外部路径长度(T))的值；通过对 h 的归纳法这可容易地证明。它的外部路径长度是 $\frac{3}{5}hF_{h-1}+\frac{4}{5}(h-1)F_h=(\phi/\sqrt{5})hF_{h+1}+O(F_{h+1})=\Theta(h\phi^h)$。因此任何 N 个节点平衡树的路径长度至多是

$$\min_h(hN-\Theta(h\phi^h)+O(N)) \leqslant N\log_\phi N - N\log_\phi \log_\phi N + O(N)$$

而且如果 N 很大而且 $k=\lceil \lg N \rceil$，$h=\lfloor k/\lg \phi - \log_{\phi^k} \rfloor = \log_\phi N - \log_\phi \log_\phi N + O(1)$，我们可以构造路径长度为 $hN+O(N)$ 的一个平衡树如下：写 $N+1=F_h+F_{h-1}+\cdots+F_{k+1}+N'=F_{h+2}-F_{k+2}+N'$，并且构造在 N' 个节点上的一个完全二叉树；然后逐次地把它同阶为 $k, k+1, \cdots, h-1$ 的斐波那契数合并[参见 R. Klein 和 D. Wood, *Theoretical Comp. Sci.* **72** (1990), 251～264]。

5. 这可以通过归纳法来证明；如果 T_N 表示被构造的树，则我们有



6. $zB_j(z)B_k(z)$ 中 z^n 的系数是 n 个节点的二叉树的个数，这株树的左子树是高度为 j 的一株平衡二叉树，而其右子树是高度为 k 的一株平衡二叉树。

7. $C_{n+1}=C_n^2+2B_{n-1}B_{n-2}$；因此如果我们设 $\alpha_0=\ln 2$，$\alpha_1=0$，以及 $\alpha_{n+2}=\ln(1+2B_{n+1}B_n/C_{n+2}^2)=O(1/B_nC_{n+2})$ 和 $\theta=\exp(\alpha_0/2+\alpha_1/4+\alpha_2/8+\cdots)$，则我们求得 $0\leqslant\theta^{2^n}-C_n=C_n(\exp(\alpha_n/2+\alpha_{n+1}/4+\cdots)-1)<1$；于是 $C_n=\lfloor\theta^{2^n}\rfloor$。关于双重指数序列的一般结果，见 *Fibonacci Quarterly* **11** (1973), 429～437。θ 的表达式迅速地收敛到下列值

$$\theta = 1.43687\ 28483\ 94461\ 87580\ 04279\ 84335\ 54862\ 92481+$$

8. 设 $b_h = B_h'(1)/B_h(1)+1$,且设 $\epsilon_h = 2B_hB_{h-1}(b_h - b_{h-1})/B_{h+1}$。则 $b_1=2$,$b_{h+1}=2b_h-\epsilon_h$ 和 $\epsilon_h = O(b_h/B_{h-1})$;因此 $b_h = 2^h\beta + r_n$,其中

$$\beta = 1 - \frac{1}{4}\epsilon_1 - \frac{1}{8}\epsilon_2 - \cdots =$$

$$0.70117\ 98151\ 02026\ 33972\ 44868\ 92779\ 46053\ 74616+$$

而且对于很大的 h 说来 $r_h = \epsilon_h/2 + \epsilon_{h+1}/4 + \cdots$ 极其小[*Zhurnal Vychisl Matem. i Matem. Fiziki* **6**,**2** (1966),389～394。E. M. Reingold,*Fib. Quart* **17** (1979),151～157 得到了对于2-3树的类似结果。]

9. Andrew Odlyzko 已经证明,平衡树的数目渐近地是

$$c^n f(\log_{(\sqrt{10}+2)/3} n)/n$$

其中 c≈1.916067 和 $f(x)=f(x+1)$。他的技术也将产生平均高度。[参见*Congressus Numerantium* **42** (1984),27～52,在这篇论文中他也讨论 2-3 树的枚举。]

10. [*Inf. Proc. Letters* **17** (1983),17～20] 设 $X_1,\cdots,X_N$ 是一些节点,并给定它们的平衡因子 B(X_k)。为构造这树,置 $k\leftarrow 0$ 并计算 TREE(∞),其中 TREE($h$max)是带有局部变量 h,h'和 Q 的下列递归过程:置 $h\leftarrow 0$,Q←Λ;然后当 $h<h$max 和 $k<N$ 时置 $k\leftarrow k+1$,$h'\leftarrow h+$B(X_k),LEFT(X_k)←Q,RIGHT(X_k)←TREE(h'),$h\leftarrow$max(h,h')$+1$,Q←X_k;返回 Q。(树 Q 有高度 h 且对应于自进入此过程以来已经读入的平衡因子。)即使当|B(X_k)|>1 时此算法仍有效。

11. 当 $n\geqslant 2$ 时显然有和－－B 和＋－B 同样多的＋A,而且在＋和－之间有对称性。如果有 M 个＋A 和－A 类型的节点,当 $n\geqslant 1$ 时所有可能情况的考虑表明,下一个随机插入得到 $M-1$ 个这样节点的概率为 $3M/(n+1)$,否则它得到 $M+1$ 个这样的节点。由此得出结果。[*SICOMP* **8** (1979),33～41;在*SICOMP* **11** (1982),748～780 中 Kurt Mchlhorn 推广了对于删除的分析。有关在这样的"次要分析"中最新发展的综述,参见 R. A. Baeza-Yates,*Computing Surreys* **27** (1995),109～119,这种分析典型地使用在习题 6.2.4-8 中说明的方法。]

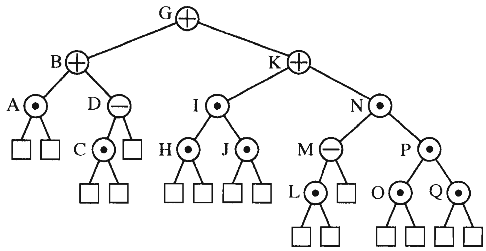
12. 当插入(12)的第二个外部节点时出现极大值;$C=4$,$C1=3$,$D=3$,$A=C2=F=G1=H1=V1=1$,总计用 $132u$ 的时间。当插入(13)的倒数第三个外节点时出现极小值;$C=2$,$C1=C2=1$,$D=2$,总计用 $61u$ 的时间。[程序 6.2.2T 相应的数字是 $74u$ 和 $26u$。]

13. 当树改变时,仅仅需要更新 $O(\log N)$个 RANK 值;"简单的"系统可能需要非常广泛的改动。

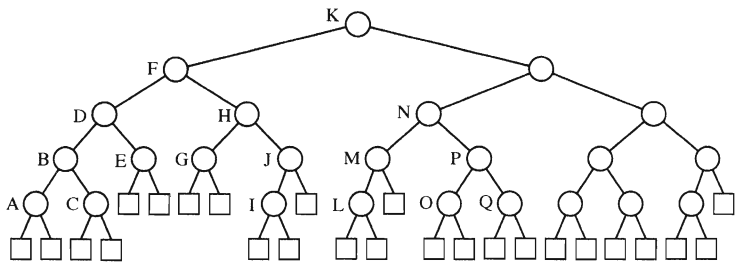
14. 是。(但是在表上的典型的操作是充分地非随机的,以致大概将出现退化树。)

15. 使用算法 6.2.2T,且在步骤 T1 中置 m 为 0,而且每当在步骤 T2 中 $K\geqslant$ KEY(P)时,$m\leftarrow m+$RANK(P)。

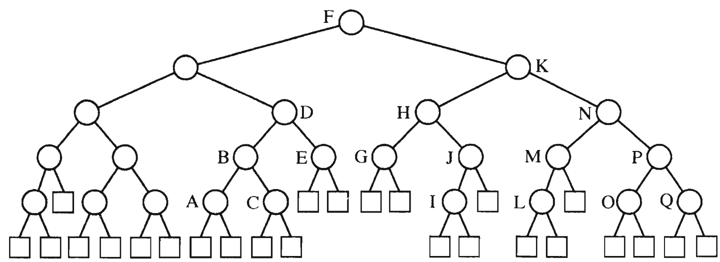
16. 删去 E;执行情况 3 在 D 处重新平衡。删去 G;以 G 代 F;执行情况 2 在 H 处重新平衡;在 K 处调整平衡因子。



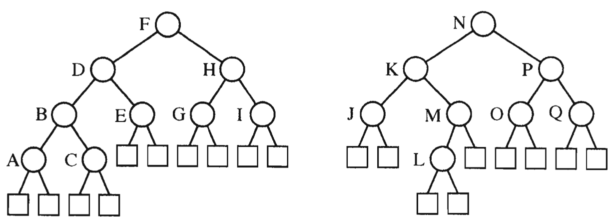
17. (a)



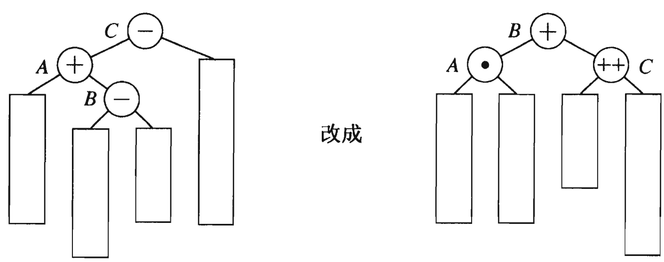
(b)



18.

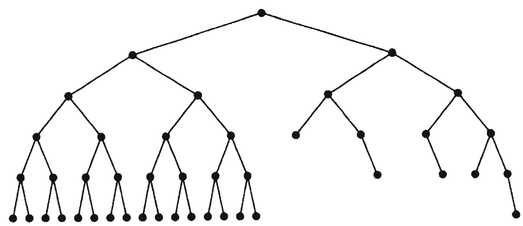


19.(由 Clark Crane 给出的解。)有一种情况,它不能由根处的单转动或双转动来处理,即是把



而后通过应用在 C 处的单转动或双转动重新解决不平衡性。

20.



也许最困难的是在这株的树的最左边插入一个新节点。但 K. J. Räihä 和 S. H. Zweben 已经设计了一个一般的算法，它花费 $O(\log N)$步。[*CACM* **22** (1979),508 ~512。]

21.算法 A 在阶为 $N\log N$ 的步骤中作此工作（见习题 5）；下列算法使用一个递归方法的有趣的迭代表达方案，在 $O(N)$步内建立同一些树。我们使用三个辅助表：

$\mathrm{D}_1,\ldots,\mathrm{D}_l$ （实际上控制递归的一个二进计数器）

$\mathrm{J}_1,\ldots,\mathrm{J}_l$ （指向交接点的指针表）

$\mathrm{T}_1,\ldots,\mathrm{T}_l$ （指向树的一个指针表）

这里 $l=\lceil\lg(N+1)\rceil$。为了方便起见，这个算法也置 $\mathrm{D}_0\leftarrow 1,\mathrm{J}_0\leftarrow\mathrm{J}_{l+1}\leftarrow\Lambda$。

G1.［初始化］ 置 $l\leftarrow 0,\mathrm{J}_0\leftarrow\mathrm{J}_1\leftarrow\Lambda,\mathrm{D}_0\leftarrow 1$。

G2.［得到下一项目］ 设 P 指向下一个输入节点。（我们可以调用另一个共行子程序以便得到 P。）如果已无输入，则转到 G5。否则，置 $k\leftarrow 1$，$\mathrm{Q}\leftarrow\Lambda$，而且交换 $\mathrm{P}\leftrightarrow\mathrm{J}_1$。

G3.［进位］ 如果 $k>l$（或者，等价地，如果 $\mathrm{P}=\Lambda$），则置 $l\leftarrow l+1,\mathrm{D}_k\leftarrow 1,\mathrm{T}_k\leftarrow\mathrm{Q},\mathrm{J}_{k+1}\leftarrow\Lambda$，并返回 G2。否则置 $\mathrm{D}_k\leftarrow 1-\mathrm{D}_k$，交换 $\mathrm{Q}\leftrightarrow\mathrm{T}_k,\mathrm{P}\leftrightarrow\mathrm{J}_{k+1}$，而且 k 增加 1。如果现在 $\mathrm{D}_{k-1}=0$，则重复这一步骤。

G4.［连接］ 置 LLINK(P)$\leftarrow\mathrm{T}_k$，RLINK(P)$\leftarrow$Q，B(P)$\leftarrow 0$，$\mathrm{T}_k\leftarrow$P，并且返回 G2。

G5.［完成］ 对于 $1\leqslant k\leqslant l$，置 LLINK(J_k)$\leftarrow\mathrm{T}_k$，RLINK(J_k)$\leftarrow\mathrm{J}_{k-1}$，B($\mathrm{J}_k$)$\leftarrow 1-\mathrm{D}_{k-1}$，然后结束这个算法；$\mathrm{J}_l$ 指向所求树的根。 ▌

步骤 G3 被执行 $2N-\nu(N)$次，其中 $\nu(N)$是 N 的二进表示中 1 的个数。

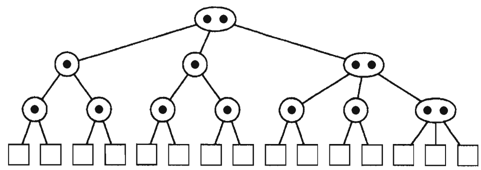
22.具有 N 个内节点的一株加权平衡树的高度，总处于 $\lg(N+1)$和 $2\lg(N+1)$之间。为了达到这个上限，注意这个根的较大的子树至多有$(N+1)/\sqrt{2}$个外节点。

23.(a)构造一株树，其右子树是具有 2^n-1 个节点的完备的二叉树，其左子树是具有 $F_{n+1}-1$ 个节点的一株斐波那契树。(b)构造一株加权平衡树其右子树大约是 $2\lg N$ 级高，其左子树大约是 $\lg N$ 级高，（参见习题 22）。

24.考虑满足这个条件但不是完全平衡的最小树。则它的左和右子树是完全平衡的，所以它们分别有 2^l 和 2^r 个外节点，其中 $l\neq r$。但这同所述条件矛盾。

25.在树的底部插入一个节点之后,我们从底往上地校验在查找通路上每个节点处权的平衡。假设在已经于右子树中插入一个新节点之后,在(1)中节点 A 处出现不平衡性,而 B 和它的子树是加权平衡的。则一次单转动将恢复平衡,除非$(|\alpha|+|\beta|)/|\gamma|>\sqrt{2}+1$,其中$|x|$表示在一个树 x 中的外节点数。但在这种情况下,可以证明一个双转动已足够[见*SICOMP* **2** (1973),33~34]。

27.有时需要在包含两个键码的节点中作两次比较。在类似于下边那样的一株树中,出现最坏的情况,它有时需要进行 $2\lg(N+2)-2$ 次比较。



29.姚期智给出的部分解:对于 $N\geqslant 6$ 个键码,最低级将平均包含$\frac{2}{7}(N+1)$个单键码节点和$\frac{1}{7}(N+1)$个双键码节点。对于很大的 N 平均的总节点数在 $0.70N$ 和 $0.79N$ 之间[*Acta Informatica* **9** (1978),159~170]。

30.对于最好适合,按大小次序排列诸记录,且以一个任意的规则来打破相等情况(参见习题 2.5-9)。对于第一个适合,按位置次序安排诸记录,而且在每个节点中以一个额外的字段指出在以该节点为根的子树中最大区域的大小。这个额外的字段在插入和删去之下仍可保持。(尽管运行时间是 $O(\log n)$,但实际上大概它仍然不能胜过习题 2.5-6 中的“ROVER”方法;但没有 ROVER 时存储分配可能更好些,因为在这种情况下通常遇紧急情况时可有一很大的空区域供使用。)

R. P. Brent,*ACM Trans. Prog. Languages and Systems* **11** (1989),388~403 提出了一个改进方法。

31.使用一个接近于平衡的树,并对于最左部分附加向上的链接,另有一个栈,栈中内容用于沿着这条通路作推迟的平衡因子校准。(每个插入作有限次这样一些校准。)

这个问题可加以推广,以要求 $O(\log m)$步来寻找,插入和/或删除在任何给定的“指针”之外 m 步的项,其中一旦找到任何键码,它就可作为后面操作的一个“指针”。[参见 S. Huddleston 和 K. Mehlhorn, *Acta Inf.* **17** (1982),157~184]。

32.每一个右转动使诸 r 之一加 1 而使其它的保持不变;因此 $r_k\leqslant r'_k$ 是必要的。为了证明它是充分的,假设对于 $1\leqslant j<k$,$r_j=r'_j$ 但 $r_k<r'_k$。于是存在一个右转动,它把 r_k 增加成一个$\leqslant r'_k$的值,因为数 $r_1r_2\cdots r_n$ 满足习题 2.3.3-19(a)的条件。

注:由 D. Tamari 于 1951 年首先引入的这个偏序,有许多有趣的性质。任何两棵树都有由右子树大小 $\min(r_1,r'_1)\min(r_2,r'_2)\cdots\min(r_n,r'_n)$,确定的最大下限

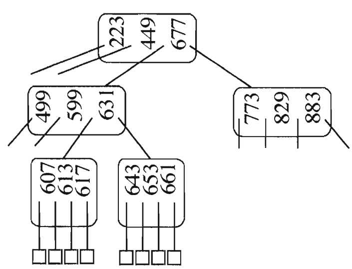
$T \wedge T'$，以及由左子树大小 $\min(l_1, l_1')\min(l_2, l_2')\cdots\min(l_n, l_n')$确定的最小上限 $T \vee T'$。当然左子树大小比算法 B 和 C 的 RANK 子段小 1。关于进一步的信息，参见 H. Friedman 和 D. Tamari，*J. Combinatorial Theory* **2** (1967)，215～242，**4**(1968)，201；C. Creene，*Europ. J. Combinatorics* **9** (1988)，225～240；D. D. Sleator，R. E. Tarjan 以及 W. P. Thurston，*J. Amer. Math. Soc.* **1** (1988)，647～681；J. M. Pallo，*Theoretical Informatics and Applic* **27** (1993)，341～348；M. K. Bennett 和 G. Birkhoff *Algebra Universalis* **32** (1994)，115～144；P. H. Edlemn 和 V. Reiner，*Mathematika* **43** (1996)，127～154。

33. 首先，我们可以在每个节点 P 中把存储减少成一个二进位的 A(P)，使得每当 LLINK(P)和 RLINK(P)都非空时，B(P) = A(RLINK(P)) − A(LLINK(P))；否则 B(P)是已知的。而且，每当 LLINK(P)和 RLINK(P)都为空时，可假定 A(P) = 0。于是每当 A(P) = 1 时通过交换 LLINK(P)和 RLINK(P)，在所有其它节点的 A(P)可删去。KEY(P)同 KEY(LLINK(P))或 KEY(RLINK(P))的比较将确定 A(P)。

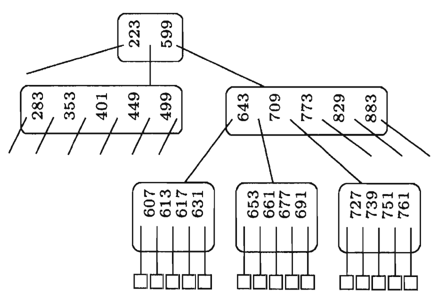
当然，在指针总是偶数的机器上，在每个节点上存在两个无用的二进位。如同在习题 2.3.1-37 中那样，进一步的节省是可能的。

6.2.4 小节

1. 双节点分开：



2. 改变的节点：



（当然一株 B^* 树没有非根的 3 键码节点，尽管图 30 有这种节点。）

3. (a) $1 + 2\cdot 50 + 2\cdot 51\cdot 50 + 2\cdot 51\cdot 51\cdot 50 = 2\cdot 51^3 - 1 = 265301$。(b) $1 + 2\cdot 50 + (2\cdot 51\cdot 100 - 100) + ((2\cdot 51\cdot 101 - 100)\cdot 100 - 100) = 101^3 = 1030301$。(c) 1 +

$2\cdot 66+(2\cdot 67\cdot 66+2)+(2\cdot 67\cdot 67\cdot 66+2\cdot 67)=601661$。(小于(b)!)

4.在分开一个非根的节点之前,先弄清楚:它确有两个已装满的兄弟,然后才把这三个节点分成为4个。若此节点是根,则仅当它有 $3\lfloor(3m-3)/4\rfloor$ 个键码以上时才能分开它。

5.解释1,试求所述极小的极大:450。(如果我们有1005个字符,而且被传送给父节点的键码必须是50个字符长,则最坏的情况就出现了:445字符+指针+50个字符的键码+指针+50个字符的键码+指针+445字符。)

解释2,试使分开之后键码的个数相等,以便保持高的分支因子:155(15个短的键码后边跟有一些16个字符长的键码)。

关于进一步的评述,请见 E. M. McCreight, *CACM* **20** (1977), 670–674。

7.如果有待删去的键码不在级 $l-1$ 上,则以它的后继者代替它,并删去这个后继者。为了删去在级 $l-1$ 上的一个键码,我们简单地抹掉它;如果这使得节点太空,则就考察它的右(或左)兄弟,并且"下溢",即,从这个兄弟移进一些键码,使得这两个节点近似地有相同的数据量。这下溢的操作仅当该兄弟的装满程度为最低时才无效,但在该种情况下两个节点可以叠合成一个(同来自它们的父节点的一个键码在一起);这样一个叠合可能引起这个父节点进一步下溢,等等。对于如同在习题5中那样的可变长的键码,当一个父节点的诸键码之一变成更长时,它可能需要分开。

8.给定有 N 个内节点的一个树 $\mathscr{T}$,并设有 $a_k^{(j)}$ 个外节点,它需要 k 个访问,且其父节点属于包含 j 个键码的一个页;并设 $A^j(z)$ 是对应的生成函数。于是 $A^{(1)}(1)+\cdots+A^{(M)}(1)=N+1$。(注意,对于 $1\leqslant j<M$, $a_k^{(j)}$ 是 $j+1$ 的倍数。)下一个随机插入导致 $N+1$ 个同等可能的树,通过某个系数 $a_k^{(j)}$ 减 $j+1$ 和加 $j+2$ 到 $a_k^{(j+1)}$ 上;或者(如果 $j=M$)某 $a_k^{(M)}$ 减1并且加2到 $a_{k+1}^{(1)}$ 上,可得到其生成函数。现在 $B_N^{(j)}(z)$ 是 $(N+1)^{-1}$ 乘以对于 $\mathscr{T}$ 的生成函数 $A^{(j)}(z)$ 取遍所有树 $\mathscr{T}$ 之和,再乘以 $\mathscr{T}$ 出现的概率;这即得出所述的递推关系。

递推式有形式

$$(B_N^{(1)}(z),\cdots,B_N^{(M)}(z))^T=(I+(N+1)^{-1}W(z))(B_{N-1}^{(1)}(z),\cdots,B_{N-1}^{(M)}(z))^T=$$

$$\cdots=g_N(W(z))(0,\cdots,0,1)^T$$

其中

$$g_n(x)=(1+\frac{x}{(n+1)})\cdots(1+\frac{x}{2})=\frac{1}{(x+1)}\binom{x+n+1}{n+1}$$

由此得出 $C'_N=(1,\cdots,1)(B_N^{(1)'}(1),\cdots,B_N^{(M)'}(1))^T=2B_{N-1}^{(M)}(1)/(N+1)+C'_{N-1}=2f_N(W)_{MM}$,其中 $f_n(x)=g_{n-1}(x)/(n+1)+\cdots+g_0(x)/2=(g_n(x)-1)/x$,而且 $W=W(1)$(下标 MM 表示矩阵的右下角元素)。现在 $W=S^{-1}\mathrm{diag}(\lambda_1,\cdots,\lambda_M)S$,

其中 S 是某个矩阵,$\mathrm{diag}(\lambda_1,\cdots,\lambda_n)$表示其元素是 $\chi(\lambda)=(\lambda+2)\cdots(\lambda+M+1)-(M+1)!$ 的根的对角矩阵。(这些根是不同的,因为 $\chi(\lambda)=\chi'(\lambda)=0$ 意味着 $1/(\lambda+2)+\cdots+1/(\lambda+M+1)=0$;后者仅当 λ 是实数且 $-M-1<\lambda<-2$ 时才能成立,它意味着$|\lambda+2|\cdots|\lambda+M+1|<(M+1)!$,矛盾。)如果 $p(x)$是任意多项式,则 $p(W)=p(S^{-1}\mathrm{diag}(\lambda_1,\cdots,\lambda_M)S)=S^{-1}\mathrm{diag}(p(\lambda_1),\cdots,p(\lambda_M))S$;因此 $p(W)$的右下角元素有形式 $c_1p(\lambda_1)+\cdots+c_Mp(\lambda_M)$,其中 $c_1,c_2,\cdots,c_M$ 是某些同 p 无关的常数。这些常数可以通过置 $p(\lambda)=\chi(\lambda)/(\lambda-\lambda_i)$来求值;由于对于 $0\leqslant k\leqslant M-1$,$(W^k)_{MM}=(-2)^k$,我们有 $p(W)_{MM}=p(-2)=(M+1)!\ /(\lambda_j+2)=c_jp(\lambda_i)=c_j\chi'(\lambda_j)=c_j(M+1)!\ (1/(\lambda_j+2)+\cdots+1/(\lambda_j+M+1))$;因此,$c_j=(\lambda_j+2)^{-1}((1/(\lambda_j+2)+\cdots+1/(\lambda_j+M+1))^{-1}$。这得出了一个"显式"公式 $C'_N=\sum_{j=1}^{M}2c_jf_N(\lambda_j)$;剩下只是研究诸根 λ_j。注意对于所有 j,$|\lambda_j+M+1|\leqslant M+1$,否则我们有$|\lambda_j+2|\cdots|\lambda_j+M+1|>(M+1)!$,矛盾。取 $\lambda_1=0$,这意味着对 $2\leqslant j\leqslant M$,$\Re(\lambda_j)<0$。由等式 1.2.5-(15),当 $n\to\infty$时 $g_n(x)\sim(n+1)^x/\Gamma(x+2)$;因此对于 $2\leqslant j\leqslant M$,$g_n(\lambda_j)\to 0$。结果 $C'_N=2c_1f_N(0)+O(1)=H_N/(H_{M+1}-1)+O(1)$。

注意:上述分析也与在 5.2.2 小节中简短地讨论的"抽样排序算法"有关。这些计算很容易推广以证明对于 $1\leqslant j<M$,$B_N^{(j)}(1)\sim(H_{M+1}-1)^{-1}/(j+2)$,且 $B_N^{(M)}(1)\sim(H_{M+1}-1)^{-1}/2$。因此在未充满的页上内节点的总数近似地为

$$\left(\frac{1}{3\times 2}+\frac{2}{4\times 3}+\cdots+\frac{M-1}{(M+1)\times M}\right)\frac{N}{H_{M+1}-1}=\left(1-\frac{M}{(M+1)(H_{M+1}-1)}\right)N$$

因此所用的页的总数近似地为

$$\left(\frac{1}{3\times 2}+\frac{1}{4\times 3}+\cdots+\frac{1}{(M-1)\times M}+\frac{1}{M+1}\right)\frac{N}{H_{M+1}-1}=\frac{N}{2(H_{M+1}-1)}$$

并产生 $2(H_{M+1}-1)/M$ 的近似的存储利用。

这个分析已由 Mahmoud 和 Pittel[*J. Algorithms* **10** (1989),52~75]作了推广,他们发现,存储利用的方差经历一个令人惊讶的阶段转换,当 $M\leqslant 25$ 时方差为 $\Theta(N)$;但当 $M\geqslant 26$ 时,它渐近地为 $f(N)N^{-1+2\alpha}$,其中如果 $-\frac{1}{2}+\alpha+\beta\mathrm{i}$ 和 $-\frac{1}{2}+\alpha-\beta\mathrm{i}$ 是有最大实部的非 0 根 λ_j,则 $f(e^{\pi/\beta}N)=f(N)$。

L. Devroye 曾分析了这样的树的高度。[*Random Structures and Algorithms* **1** (1990),191~203];也见 B. Pittel, *Random Structures and Algorithms* **5** (1994),337~347。

9. 是;例如,我们可以以 i 代替(1)中的每个 K_i 加上在子树 $P_0,\ldots,P_{i-1}$中键码的个数。可适当地修改查找,插入和删去算法。

10. 简单的概述,推广页方案,使得在一个时刻对一个用户赋予对缓冲区的排它访问;必须小心地修改查找,插入和删去算法使得仅仅对于绝对需要时的一个极限

时间。这样的排它访问才被批准，并且以不出现死锁这样一种方式进行。关于细节，请见 B. Samadi, *Inf. Proc. Letters* **5** (1976), 107～112；R. Bayer 和 M. Schkolnick, *Acta Inf.* **9** (1977), 1～21；Y. Sagiv, *J. Comp. Syst. Sci.* **33** (1986), 275～296。

6.3 节

1. 叶(复数)。*

2. 利用新的键码作为变元实施算法 T；它将在步骤 T3 或 T4 中以失败告终。如果是在 T3 中，则只需置 NODE(P)的表项 k 成为 K 并终止此插入算法。否则置这个表项成为一新节点 Q⇐AVAIL 的地址，该节点只包含空的链接，然后置 P←Q。现在置 k 和 k' 分别成为 K 和 X 的下一个字符；如果 $k \neq k'$，则把 K 存入 NODE(P)的位置 k 处，并把 X 存入位置 k' 中，但如果 $k = k'$，则再一次使 k 的位置指向一个新节点 Q⇐AVAIL，置 P←Q，并且重复这个过程直到最终 $k \neq k'$ 为止。(我们必须假定没有一个键码是另一个的前缀。)

3. 在键码出现的节点处，以一个空的链接来代替此键码。如果这个节点现在成了无用的，因为它除了有一个项是键码 X 外，其它的所有项都是空的，则删去这个节点，并且以 X 代替它父节点中相应的指针。如果父节点现在已成为无用，则以同样的方式删去它。

4. 在压缩了的表中，如果查找是成功的，则情况和完全的表是一样的，但如果查找是不成功的，则可能要多费若干次另外的迭代。例如，如像 TRASH 这样的输入变元将使程序 T 花费六次迭代(多于五次!)；这是最坏的情况。有必要验证，对空白的序列不可能有无穷的循环。(这个有名的 49 位压缩法是由 J. Scot Fishburn 给出的，他证明了 48 个位置是不够的。)

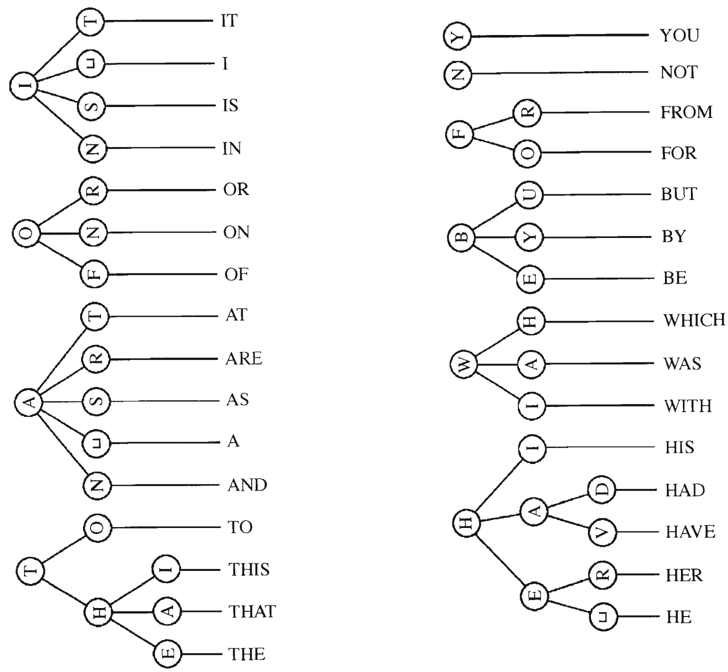
Kurt Maly, *CACM* **19** (1976), 409～415 已经提出对于检索结构存储的一个较慢的但有更多种多样节省的方法。

一般来说，如果我们要来压缩分别含 $x_1, \cdots, x_n$ 个非 0 项的 n 个稀疏表，则用同以前放置的表不相冲突并以极小数量 r_j 补偿第 j 个表的“最先适配”方法，将有 $r_j \leqslant (x_1 + \cdots + x_{j-1})x_j$，因为每个以前的非 0 项顶多能封锁 x_j 个偏离。对于表 1 中的数据这最坏情况估计给出 $r_j \leqslant 93$，并保证分别含有 10, 5, 4, 3, 3, 3, 3, 3, 2, 2, 2, 2 个非 0 项、长度为 30 的任何 12 个表可以压缩成 93 + 30 个连续的单元，而不论非 0 项的模式如何。R. E. Tarjan 和姚期智已经给出对于这个方法进一步的改进，*CACM* **22** (1979), 606～611。由 F. M. Liang 给出的压缩检索结构的一个动态实现，用于 T_EX 打字系统的连字表格中；参见 D. E. Knuth, *CACM* **29** (1986), 471～478；*Literate Programming* (1992), 206～233。

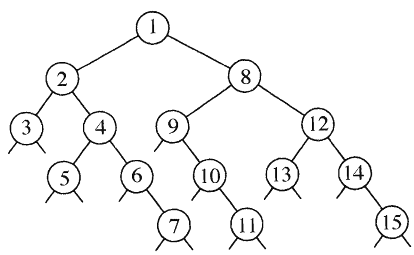
5. 在每个族中，通过从左到右地以概率递减的次序排列字母，首先测试最可能的结果。这排列的最优性可以如同在定理 6.1S 中那样证明。[参考 *CACM* **12**

* 树(tree)的叶为 leaf，因此检索结构(trie)的叶为 Lief。前者复数为 Leaves，后者复数为 Lieves。

(1969),72～76。]



6.



7.例如,8,4,1,2,3,5,6,7,12,9,10,11,13,14,15。(不管使用什么序列,左子树在级 4 上不能包含多于两个节点,右子树也不能。)甚至这株“最坏”的树也是在最好的四株树之内,所以我们看到,数字查找树对于插入次序不是非常敏感。

8.是。KEY 字段现在仅仅包含一个截断了的键码;由节点位置所蕴涵的诸前导二进位被砍掉了。(算法 T 的一项类似的修改是可能的。)

9.

START	LDX	K	1	<u>*D1.初始化*。</u>($rX\equiv K$)
	LD1	ROOT	1	P←ROOT($rI1\equiv P$)
	JMP	2F	1	

```
4H    LD2    0,1(RLINK)   C2     D4. 右移, Q←RLINK(P)
      J2Z    5F           C2     如果 Q=Λ,则移到 D5
1H    ENT1   0,2          C-1    P←Q
2H    CMPX   1,1          C      D2. 比较
      JE     SUCCESS      C      如果 K=KEY(P),则转出
      SLB    1            C-S    把 K 左移一个二进位
      JAO    4B           C-S    如果移掉的二进位是 1,则转到 D4
      LD2    0,1(LLINK)   C1     D3. 左移。Q←LLINK(P)
      J2NZ   1B           C1     如果 Q≠Λ,则以 P←Q 转到 D2
5H    如同在程序 6.2.2T 中那样继续,并交换 rA 和 rX 的作用。 ▌
```

这个程序查找阶段的运行时间是$(10C-3S+4)u$,其中 $C-S$ 是二进位探查的数目。因此,对于随机数据,近似的平均运行时间是

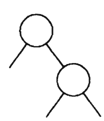
	成功的	不成功的
程序 6.2.2T	$15\ln N-12.34$	$15\ln N-2.34$
这个程序	$14.4\ln N-6.17$	$14.4\ln N+1.26$

(因此,除非 N 非常大;程序 6.2.2T 要稍许快些。)

10. 设$\oplus$表示对于 n 个二进位数的异或操作,而且设 $f(x)=n-\lceil\lg(x+1)\rceil$,是 x 的前导 0 位的个数。一种解法:(b) 如果算法 T 执行的一个查找在步骤 T3 处以失败告终,则 K 比至今所作的二进位探查数小 1;否则如果查找在步骤 T4 处结束,则 $k=f(K\oplus X)$。(a,c) 做正规的查找,但是也要随时记住相对于在查找过程中与 K 作过比较的所有 KEY(P),$K\oplus$KEY(P)的极小值 x。于是 $k=f(x)$。(证明,和 K 比较过的键码与 K 相同的二进位个数,比没有同 K 比较过的其它键码为多。在情况(a)中,极大的 k 值或者对于$\leqslant K$ 的最大键码,或者对于$>K$ 的最小键码出现。)

11. 否;消去只有一个空子树的一个节点会"忘掉"在非空子树键码中的一个二进位。为删去一个节点,应该以它的终端后代之一代替它,比如说,只要可能时通过向右查找来实现。

12. 把在 0 和 1 之间的三个随机数 α,β,γ 插入到开始时为空的一株树中;然后使用前面答案中所建议的算法,以概率 p 删去α,以概率 q 删去β,以概率 r 删去γ,则我们以概率$\frac{1}{4}p+\frac{1}{2}q+\frac{1}{2}r$ 得到树



而且仅当 $p=0$ 时,这概率是$\frac{1}{2}$。

13. 对每个节点加一个 KEY 字段,而且在考察步骤 T2 的向量元素之前把 K 同这个键码加以比较。表 1 将改变如下:节点(1),…,(12)将分别包含键码 THE,AND,

BE,FOR,HIS,IN,OF,TO,WITH,HAVE,HE,THAT(如果我们按频率递减的次序插入它们的话),而且这些键码将从它们以前的位置中被删去。[对应的程序在这种情况下将比程序 T 更慢和更复杂。算法 D 的一个更直接的 M 进推广将建立具有 N 个节点的一株树,每个节点有一个键码和 M 个链接]。

14. 如果 $j \leqslant n$,则仅有一个位置,即 KEY(P)。但是如果 $j > n$,则通过遍历节点 P 的子树找出所有出现的集合:如果有 r 个出现,则这株子树包含 $r-1$ 个节点(包括节点 P 在内),而且因此它有 TAG = 1 的 r 个链接字段的;这些链接字段都指向访问和 K 匹配的 TEXT 的位置的所有节点(完全不必再检验 TEXT 了)。

15. 为开始形成这株树,置 KEY(HEAD)成为第一个 TEXT 访问,而且置 LLINK(HEAD)←HEAD,LTAG(HEAD)←1。用下列插入算法把其余的 TEXT 访问记入树中。

置 K 成为我们希望记入的新键码。(这是插入算法所作的对于 TEXT 阵列的第一次访问)。实施算法 P;它必然以失败告终,因为不允许一个键码是另外键码的一个前缀。(步骤 P6 作对于 TEXT 的第二次访问;不再需要更多的访问了!)现在假设在步骤 P6 中找到的键码同变元 K 在前 l 个二进位中一致,但是在第 $l+1$ 个位置却不同,在 K 中是数字 b 而在键码中是 $1-b$。(尽管算法 P 中的查找可能要使 j 比 l 大得多,但可以证明,这里所确定的过程将在 K 和现存的任何键码之间求得最长的匹配。于是,以 K 的前 l 位开始的正文的所有键码,都有 $1-b$ 作为它的第 $l+1$ 位二进位。)现在重复算法 P 并且以这些 l 个前导二进位代替 K(即 $n \leftarrow l$)。这次查找将是成功的,所以我们不必实施步骤 P6。现在置 R⇐AVAIL,KEY(R)←TEXT 中新键码的位置。如果 LLINK(Q) = P,则置 LLINK(Q)←R,t←LTAG(Q),LTAG(Q)←0;否则置 RLINK(Q)←R,t←RTAG(Q),RTAG(Q)←0。如果 $b=0$,置 LTAG(R)←1,LLINK(R)←R,RTAG(R)←t,RLINK(R)←P;否则置 RTAG(R)←1,RLINK(R)←R,LTAG(R)←t,LLINK(R)←P。如果 $t=1$,则置 SKIP(R)←$1+l-j$;否则置 SKIP(R)←$1+l-j+$SKIP(P)和 SKIP(P)←$j-l-1$。

16. 树的建立恰恰需要从下边的一个节点到该节点的一条虚线的链接;它来自树的一个部分,在这部分中此键码第一次不同于所有其它键码。如果树中没有这样的部分,则这些算法失败。我们可以简单地去掉作为其它键码的前缀的键码,但那样的话习题 14 的算法将没有足够的数据来找出这个变元的所有出现。

17. 如果我们定义 $a_0 = a_1 = 0$,则

$$x_n = a_n + \sum_{k \geqslant 2} \binom{n}{k} (-1)^k \hat{a}_k / (m^{k-1} - 1) = \sum_{k \geqslant 2} \binom{n}{k} (-1)^k \hat{a}_k m^{k-1} / (m^{k-1} - 1)$$

18. 为了解(4),我们需要 $a_n = [n > 1]$,即 $\hat{a}_n = [n=0] - 1 + n$;因此对于 $N \geqslant 2$,我们得到 $A_N = 1 - U_N + V_N$,这里用的是习题 19 的记号,其中 $U_N = K(N, 0, M)$ 和 $V_N = K(N, 1, M)$。类似地,为了解(5),取 $a_n = n - [n=1] = \hat{a}_n$,而且对于 $N \geqslant 2$,得到 $C_N = N + V_N$。

19. 对于 $s=1$,我们有 $V_n = K(n, 1, m) = n(\ln n + \gamma)/\ln m - \dfrac{1}{2} - \delta_0(n-1))$

$+O(1)$,而且,对于 $s\geqslant 2$,我们有

$$K(n,s,m)=(-1)^s n(1/\ln m+\delta_{s-1}(n-s))/s(s-1)+O(1)$$

其中

$$\delta_{s(n)}=\frac{2}{\ln m}\sum_{k\geqslant 1}\Re(\Gamma(s-2\pi ik/\ln m)\exp(2\pi ik\log_m n))$$

是 $\log n$ 的一个周期函数。[在这个推导中我们有

$$K(n+s,s,m)/(-1)^s\binom{n+s}{s}=\frac{n^{-s+1}}{2\pi i}\int_{1/2-i\infty}^{1/2+i\infty}\frac{\Gamma(z)n^{s-1-z}dz}{m^{s-1-z}-1}+O(n^{-s})$$

对于小的 m 和 s,诸 δ 小得可以忽略;参见习题 5.2.2-46。注意,对于固定的 α,$\delta_s(n-\alpha)=\delta_s(n)+O(n^{-1})$]。

20.对于(a),令 $a_n=[n>s]=1-\sum_{k=0}^{s}[n=k]$;对于(b),令 $a_n=n-\sum_{k=0}^{s}k[n=k]$;对于(c),我们要解递推式

$$y_n=\begin{cases}m^{1-n}\sum_k\binom{n}{k}(m-1)^{n-k}y_k & \text{对于 } n>s\\ \binom{n+1}{2} & \text{对于 } n\leqslant s\end{cases}$$

置 $x_n=y_n-n$ 得出习题 17 形式的递推式,且

$$a_n=(1-M^{-1})\sum_{k=0}^{s}\binom{k}{2}[n=k]$$

因此,在前边习题的记号下,答案是(a) $1-K(N,0,M)+K(N,1,M)-\cdots+(-1)^{s-1}K(N,s,M)=N/(s\ln M)-N(\delta_{-1}(N)+\delta_0(N-1)+\delta_1(N-2)/2\cdot 1+\cdots+\delta_{s-1}(N-s)/s(s-1))+O(1)$;(b) $N^{-1}(N+K(N,1,M)-2K(N,2,M)+\cdots+(-1)^{s-1}sK(N,s,M))=(\ln N+\gamma-H_{s-1})/\ln M+\frac{1}{2}-(\delta_0(N-1)+\delta_1(N-2)/1+\cdots+\delta_{s-1}(N-s)/(s-1))+O(N^{-1})$;(c) $N^{-1}(N+(1-M^{-1})\sum_{k=2}^{s}(-1)^k\binom{k}{2}K(N,k,M))=1+\frac{1}{2}(1-M^{-1})((s-1)/\ln M+\delta_1(N-2)+\cdots+\delta_{s-1}(N-s))+O(N^{-1})$。

21.设共有 A_N 个节点。非空指针的个数是 A_N-1,而且非指针的个数是 N,因此空指针的总数是 MA_N-A_N+1-N。除以 m,即得它们在任何固定位置中空指针的平均值。[A_N 的平均值出现于习题 20(a)中。]

22.对于 M^l 个前导二进位序列中的每一个,存在一个节点使得至少两个键码有这个二进位型。由于恰恰 k 个键码有一个特定的二进位形式的概率是

$$\binom{N}{k}M^{-lk}(1-M^{-l})^{N-k}$$

所以在 l 级上检索结构的平均节点数是 $M^l(1-(1-M^{-l})^N)-N(1-M^{-l})^{N-1}$。

23.更一般地说,考虑如同习题 20 中那样任意 s 的情况。如果有 a_l 个节点在

级 l 上,则它们包含 a_{l+1} 个链接以及 $Ma_l - a_{l+1}$ 个位置,在这些位置上查找可能是不成功的。数字探查的平均数因此将是 $\sum_{l\geqslant 0}(l+1)M^{-l-1}(Ma_l - a_{l+1}) = \sum_{l\geqslant 0} M^{-l}a_l$。把 a_l 的这一公式使用于一随机检索结构中,它等于

$$1 + \frac{K(N+1,1,M) - 2K(N+1,2,M) + \cdots + (-1)^s(s+1)K(N+1,s+1,M)}{N+1}$$

$$= \frac{\ln N + \gamma - H_s}{\ln M} + \frac{1}{2} - \delta_0(N) - \frac{\delta_1(N-1)}{1} - \cdots\cdots \frac{\delta_s(N-s)}{s} + O(N^{-1})$$

24.我们必须解递推式 $x_0 = x_1 = y_0 = y_1 = 0$,对于 $n\geqslant 2$,

$$x_n = m^{-n}\sum_{n_1+\cdots+n_m=n}\binom{n}{n_1,\cdots,n_m}\Bigl(x_{n_1} + \cdots + x_{n_m} + \sum_{1\leqslant j\leqslant m}[n_j \neq 0]\Bigr) =$$

$$a_n + m^{1-n}\sum_k \binom{n}{k}x_k$$

$$y_n = m^{-n}\sum_{n_1+\cdots+n_m=n}\binom{n}{n_1,\cdots,n_m}\Bigl(y_{n_1} + \cdots + y_{n_m} + \sum_{1\leqslant i<j\leqslant m}[n_i \neq 0]n_j\Bigr) =$$

$$b_n + m^{1-n}\sum_k \binom{n}{k}y_k$$

其中,$a_n = m(1-(1-1/m)^n)$ 和 $b_n = \frac{1}{2}(m-1)n(1-(1-1/m)^{n-1})$。由习题 17 和 18,答案是(a) $X_N = N + V_N - U_N - [N=1] = A_N + N - 1$(这是可以直接得到的一个结果,因为在森林中的节点个数总比在相应的检索结构中的节点个数多 $N-1$ 个!);而(b) $y_N/N = \frac{1}{2}(M-1)V_N/N = \frac{1}{2}(M-1)(\ln N + \gamma)\ln M - \frac{1}{2} - \delta_0(N-1)) + O(N^{-1})$。

25.(a) 设 $A_N = M(N-1)/(M-1) - E_N$;于是对于 $N\geqslant 2$,我们有 $(1-M^{1-N})E_N = m - 1 - M(1-1/M)^{N-1} + M^{1-N}\sum_{0<k<N}\binom{N}{k}(M-1)^{N-k}E_k$。由于 $M-1\geqslant M(1-1/M)^{N-1}$,由归纳法,我们有 $E_N\geqslant 0$。(b)由定理 1.2.7A 连同 $x = 1/(M-1)$ 和 $n = N-1$,我们求得 $D_N = a_N + M^{1-N}\sum_k\binom{N}{k}(M-1)^{N-k}D_k$,其中 $a_1 = 0$,且对于 $N\geqslant 2, 0 < a_N < M(1-1/M)^N/\ln M \leqslant (M-1)^2/M\ln M$。因此,$0\leqslant D_N \leqslant (M-1)^2A_N/M\ln M \leqslant (M-1)(N-1)/\ln M$。

26.在习题 5.1.1-16 的第二个恒等式中取 $q = \frac{1}{2}$,$z = -\frac{1}{2}$,我们得到 $1/3 - 1/(3\cdot 7) + 1/(3\cdot 7\cdot 15) - \cdots = 0.28879$;若使用 $z = -\frac{1}{4}$ 和取这个结果的一半则要稍快些。或者,可以使用来自习题 5.1.1-14 的欧拉公式,它仅涉及 2 的负的次幂。(John Wrench 计算了 40 位的值即 0.2887880950 86602 42127 88997 21929 23078

00889+。)

27.(为了开心,下列推导进行到 $O(N^{-1})$。)在习题 5.2.2-38 和 5.2.2-48 的记号下,我们有

$$\bar{C}_N = U_N + N - 1 + \frac{V_{N+1}}{N+1} - \alpha N - \beta + \sum_{n\geqslant 2}(-1)^n 2^{-n(n+1)/2} \frac{\sum_{m\geqslant 0}(2^{1-n})^m(1-2^{-m})^N}{\prod_{r=1}^{n}(1-2^{-r})}$$

其中

$$\alpha = 2/(1\cdot 1) - 4/(3\cdot 3\cdot 1) + 8/(7\cdot 7\cdot 3\cdot 1) - 16/(15\cdot 15\cdot 7\cdot 3\cdot 1) + \cdots \approx 1.60669\ 51524\ 15291\ 76378\ 33015\ 23190\ 92458\ 04806 -$$

以及 $\beta = 2/(1\cdot 3\cdot 1) - 4/(3\cdot 7\cdot 3\cdot 1) + 8(7\cdot 15\cdot 7\cdot 3\cdot 1) - \cdots \approx 0.60670$。这个数值计算提示 $\alpha = \beta + 1$,这是不难证明的一个事实而且,α 原来竟和在 5.2.3-(19)中十分不同地定义的常数相同;参见 Karl Dilcher, *Disc. Math.* **145** (1995),83~93。由习题 5.2.2-46,$\sum_{m\geqslant 0}(2^{1-n})^m(1-2^{-m})^N$ 的值是 $O(N^{1-n})$;$V_{N+1}/(N+1) = U_{N+1} - U_N$。因此由习题 5.2.2-50,$\bar{C}_N = U_{N+1} - (\alpha - 1)N - \alpha + O(N^{-1}) = (N+1)\lg(N+1) + N((\gamma - 1)/\ln 2 + \frac{1}{2} - \alpha + \delta_{-1}(N)) + \frac{1}{2} - 1/\ln 4 - \alpha - \frac{1}{2}\delta_1(N) + O(N^{-1})$。

Kirschenhofer, Prodinger 和 Szpankowski, *SICOMP* **23** (1994),598~616 计算了一个数字查找树的内部路径长度的方差。

28.如果我们在显然的位置中以 M 替换 2,则正文中和习题 27 中的推导可应用于一般的 $M\geqslant 2$。因此在一次随机的成功查找中平均的数字探查次数是 $\bar{C}_N/N = U_{N+1} - \alpha_M + 1 + O(N^{-1}) = \log_M N + (\gamma - 1)/\ln M + \frac{1}{2} - \alpha_M + \delta_{-1}(N) + (\log_M N)/N + O(N^{-1})$;而且对于不成功的情况它是 $\bar{C}_{N+1} - \bar{C}_N = V_{N+2}/(N+2) - \alpha_M + 1 + O(N^{-1}) = \log_M N + \gamma/\ln M + \frac{1}{2} - \alpha_M - \delta_0(N+1) + O(N^{-1})$。这里 $\delta_s(n)$ 在题 19 中定义,而 $\alpha_M = \sum_{j\geqslant 0}(-1)^j M^{j+1}/(M^{j+1}-1)^2(M^j-1)\cdots(M-1)$。

29.Flajolet 和 Sedgewick [*SICOMP* **15** (1986),748~767]证明了,当 $M=2$ 时这样节点的近似平均个数是 $.372N$,而当 $M=16$ 时是 $.689N$。也请参见 Flajolet 和 Richmond 的推广, *Random Structures and Algorithms* **3** (1992),305~320。

30.通过迭代递推式,$h_n(z)$是形如

$$\binom{n}{p_1}\left(\frac{z}{2^{p_1}-1}\right)\binom{p_1}{p_2}\left(\frac{z}{2^{p_2}-1}\cdots\frac{z}{2^{p_m}-1}\right)\binom{p_m}{1},\ 对于\ n > p_1 > \cdots > p_m > 1,$$

的所有可能的项之和。

31.$h_n'(1) = V_n$。参见习题 5.2.2-36(b)。[关于 Patricia 树的 M 叉推广的方差和极限分布,参见 P. Kirschenhofer 和 H. Prodinger, *Lecture Notes in Comp. Sci.* **226**

(1986),177～185;W. Szpankowski,*JACM* **37** (1990),691～711;B. Rais,P. Jacquet 和 W. Szpankowski,*SIAM J. Discrete Math* **6** (1993),197～213。]

32. SKIP 字段之和是在对应的二进检索结构中的节点数,所以答案是 A_N(参见习题 20)。

33. (18)是这样发现的:$A(2z)-2A(z)=e^{2z}-2e^z+1+A(z)(e^z-1)$可以被变换成为 $A(2z)/(e^{2z}-1)=(e^z-1)/(e^z+1)+A(z)/(e^z-1)$。因此 $A(z)=(e^z-1)\sum_{j\geqslant 1}(e^{z/2^j}-1)/(e^{z/2^j}+1)$。现在如果 $f(z)=\sum c_nz^n$,则 $\sum_{j\geqslant 1}f(z/2^j)=\sum c_nz^n(2^n-1)$。在这种情况下 $f(z)=(e^z-1)/(e^z+1)=\tanh(z/2)$,它等于 $1-2z^{-1}(z/(e^z-1)-2z/(e^{2z}-1))=\sum_{n\geqslant 1}B_{n+1}z^n(2^{n+1}-1)/(n+1)!$。从这个公式出发,下一步该怎么走就清楚了。

34. (a)考虑 $\sum_{j\geqslant 1}\sum_{k=2}^{n-1}\binom{n}{k}B_k/2^{j(k-1)}$;由习题 1.2.11.2－4,$1^{n-1}+\cdots+(m-1)^{n-1}=(B_n(m)-B_n)/n$。(b) 设 $S_n(m)=\sum_{k=1}^{m-1}(1-k/m)^n$ 和 $T_n(m)=1/(e^{n/m}-1)$。如果 $k<m/2$,我们有 $e^{-kn/m}>\exp(n\ln(1-k/m))>\exp(-kn/m-k^2n/m^2)>e^{-kn/m}(1-k^2/m^2)$,因此 $(1-k/m)^n=e^{-kn/m}+O(e^{-kn/m}k^2n/m^2)$。由于 $S_n(m)=\sum_{k=1}^{m/2}(1-k/m)^n+O(2^{-n})$和 $T_n(m)=\sum_{k=1}^{m/2}e^{-kn/m}+O(e^{-n/2})$,因此我们有 $S_n(m)=T_n(m)+O(e^{-n/m}n/m^2)$。$O(\exp(-n/2^j)n/2^{2j})$之和是 $O(n^{-1})$,因为对于 $j\leqslant \lg n$ 这个和有 $n^{-1}(1+2/e+(2/e)^2+\cdots)$的阶,而对于 $j\geqslant \lg n$ 这个和有 $n^{-1}(1+1/4+(1/4)^2+\cdots)$的阶。(c) 当$|x|<2\pi$ 时和在 5.2.2 小节一样进行论证,然后使用解析的连续性。(d) $\frac{1}{2}\lg(n/\pi)+\gamma/(2\ln 2)-\frac{3}{4}+\delta(n)+2/n$,其中

$$\delta(n)=(2/\ln 2)\sum_{k\geqslant 1}\Re(S(-2\pi ik/\ln 2)\Gamma(-2\pi ik/\ln 2)\exp(2\pi ik\lg n))=$$
$$(1/\ln 2)\sum_{k\geqslant 1}\Re(S(1+2\pi ik/\ln 2)\exp(2\pi ik\lg(n/\pi)))/\cos h(\pi^2k/\ln 2)$$

W. Szpankowski, *JACM* **37** (1990),691～771 计算了方差和更高的矩量。

35. 诸键码必须是$\{\alpha 0\beta 0\omega_1,\alpha 0\beta 1\omega_2,\alpha 1\gamma 0\omega_3,\alpha 1\gamma 1\delta 0\omega_4,\alpha 1\gamma 1\delta 1\omega_5\}$,其中 α,β,…是 0 和 1 的串且有$|\alpha|=a-1$,$|\beta|=b-1$,等等。五个随机键码有这个形式的概率是 $5!\ 2^{a-1+b-1+c-1+d-1}/2^{a+b+a+b+a+c+a+c+d+a+c+d}=5!\ /2^{4a+b+2c+d+4}$。

36. 设有 n 个内节点。(a)$(n!\ /2^I)\prod(1/s(x))=n!\ \prod(1/2^{s(x)-1}s(x))$,其中 I 是此树的内部路径长度。(b) $((n+1)!\ /2^n)\prod(1/(2^{s(x)}-1))$。(考虑对于所有的 $a,b,c,d\geqslant 1$ 来对习题 35 的答案求和。)

37. 最小的修改了的外部路径长度实际上是 $2-1/2^{N-2}$,而且它仅出现在一株退化的树(其外部路径长度为极大者)中。[可以证明,最大的修改了的外部路径长度出现的条件是当且仅当外节点出现在至多两个相邻的级上!但是如果说凡是其外部路径长度小于另一株树的外部路径长度者,一定有一个更大的修改了的外部路径长度,这一点就不见得总是正确的了。]

38. 把求具有参数(α,β)，$\left(\alpha,\frac{1}{2}\beta\right)$，$\cdots$，$(\alpha,2^{k-n}\beta)$的那些 k 节点的树看作是一些子问题。

39. 见 Miyakawa, Yuba, Sugito 及 Hoshi Mamoru, *SICOMP* **6** (1977), 201～234。

40. 设 N/r 是这个序列的真的周期长度。构造一株 Patricia 类型的树，以 $a_0a_1\cdots$作为 TEXT，且有从位置 $0,1,\cdots,N/r-1$ 处开始的 N/r 个键码。(没有任何键码是另一个的前缀，这是由于我们对 r 的选择所致。)在每个节点中还包括一个 SIZE 字段，该字段包含在此节点下面的子树中带标志的链接字段的数目。为做此特定的操作，使用算法 P；如果这次查找是不成功的，则答案为 0，但如果它是成功的，而且 $j\leqslant n$，则答案为 r。最后，如果它是成功的，而且 $j>n$，则答案为 $r\cdot$SIZE(P)。

43. 预期的高度近似于$(1+1/s)\log_M N$，而方差为 $O(1)$。参见 H. Mendelson, *IEEE Transactions* **SE-8**(1982), 611～619；P. Flajolet, *Acta. Informatica* **20** (1983), 345～369；L. Devroye, *Acta Informatica* **21** (1984), 229～237；B. Pittél, *Advancedsin Applied Probability* , **18** (1986), 139～155；W·Szpankowski *Algorithmica* **6** (1991), 256～277。

具有 $M=2$ 的一个随机数字查找树的平均高度是近似于 $\lg n+\sqrt{2\lg n}$[Aldous 和 Shields, *Probability Theory and Related Fields* **79** (1988), 509～542]，而且对于一个随机 Patricia 树同样的结论成立。[Pittel 和 Rubin, *Journal of Combinatorial Theory* **A55** (1990), 292～312。]

44. 参见*SODA* **3** (1997), 360～369；这个查找结构同在习题 5.2.2-30 的答案中讨论的多重键码快速排序算法密切有关。J. Clément, P. Flajolet 和 B. Vallée 已经证明，三叉表示使检索结构查找大约比(2)的二叉表示快三倍，相对于访问的节点而言[参见*SODA* **9** (1998), 531～539。]

45. {THAT, THE, THIS} 在 {BUILT, HOUSE, IS, JACK} 之前，{HOUSE, IS, JACK} 在 {BUILT} 之前，{HOUSE, IS} 在 {JACK} 之前，{IS} 在 {HOUSE} 之前，{THIS} 在 {THAT, THE} 之前，以及 {THE} 在 {THAT} 之前的概率是$\frac{3}{7}\cdot\frac{3}{4}\cdot\frac{2}{3}\cdot\frac{1}{2}\cdot\frac{1}{3}\cdot\frac{1}{2}=\frac{1}{56}$。

6.4 节

1. $-37\leqslant$rI1$\leqslant 46$。因此在 TABLE 之前和之后的单元必须保证不包含同任何给定的变元相匹配的数据。例如，它们的第一个字节可以为 0，在这个范围内存储 K 肯定是坏的！[因此，在某种意义下，我们可以说，习题 6.3-4 中的方法使用较少的空间，因为该表的边界决不被超过。]

2. TOW。[读者能否找到至多 5 个字母的十个“常用的”字，填满 -10 和 30 之间所有剩下的间隔？]

3. 字符代码 A + T = I + N 和 B − E = O − R，所以我们有 f(AT) = f(IN 或 f(BE) = f(OR)。注意表 1 的指令 4 和 5 相当好地解决了这个问题，同时保持 rI1 没有太宽的

范围。

4.考虑具有 k 个对的情况,使得对于 $m=365$

$$m^{-n}n!\sum_k\binom{m}{n-k}\binom{n-k}{k}2^{-k}<\frac{1}{2}$$

最小的 n 是 88。如果你邀请 88 人(包括你自己在内),则生日试验的机会是.511065,但如果来了 87 人,则它就降低到.499455。参见 C. F. Pinzka, *AMM* **67** (1960)830。

5.散列函数是坏的,因为它假定顶多有 26 个不同的值,而且它们中的某些要比其它的那些更经常地出现。甚至对于双重散列(比如说设 $h_2(K)=1$ 加上 K 的第二个字节,而且比如说,$M=101$),查找所减慢的时间将超过由更快的散列而节省的时间。还有 $M=100$ 太小,因为 FORTRAN 程序通常都有多于 100 个不同的变量。

6.在 MIX 上不行,因为算术溢出几乎总出现(被除数太大)。[最好能计算(wK) mod M,特别是如果线性探查以 $c=1$ 被使用时,但不幸的是大多数计算机都不允许这一点,因为商溢出了。]

7.如果 $R(x)$ 是 $P(x)$ 的一个倍式,则对于所有的 $j\in S$,$R(\alpha^j)=0$ 在 $GF(2^k)$ 中。设 $R(x)=x^{a_1}+\cdots+x^{a_s}$,其中 $a_1>\cdots>a_s\geqslant 0$ 且 $s\leqslant t$,并且选择 $t-s$ 个进一步的值 $a_{s+1},\cdots,a_t$,使得 $a_1,\cdots,a_t$ 是小于 n 的不同的非负整数,Vandermonde 矩阵

$$\begin{pmatrix}\alpha^{a_1}\cdots\alpha^{a_t}\\ \alpha^{2a_1}\cdots\alpha^{2a_t}\\ \vdots\qquad\vdots\\ \alpha^{ta_1}\cdots\alpha^{ta_t}\end{pmatrix}$$

是奇异的,因为它的前 s 个列之和为 0。但这同 $\alpha^{a_1},\cdots,\alpha^{a_t}$ 是 $GF(2^k)$ 的不同元素这一事实矛盾。(见习题 1.2.3-37。)

[多项式散列的思想由 M. Hanan, S. Muroga, F. P. Palermo, N. Raver 及 G. Schay 所首创;见*IBM J. Research & Development* **7** (1963),121~129;*U. S. Patent* **3311888** (1967)。]

8.由归纳法。强归纳假设可通过对于 $0\leqslant r\leqslant a_k$,$\{(-1)^k(rq^k+q_{k-1})\theta\}=(-1)^k(r(q_k\theta-p_k)+(q_{k-1}\theta-p_{k-1}))$这一事实而得到补充。对于 $n=q_1,q_2+q_1,2q_2+q_1,\cdots,a_2q_2+q_1=0q_4+q_3,q_4+q_3,\cdots,a_4q_4+q_3=0q_6+q_5,\cdots$出现$\{n\theta\}$的"创记录地低"的值;对于 $n=q_0,q_1+q_0,\cdots,a_1q_1+q_0=0q_3+q_2,\cdots$出现"创记录地高"的值。这些是当形成一个具有新的长度编号为 0 的区间时的一些步骤。[进一步的结构可以通过推广习题 1.2.8-34 的斐波那契数系而推演出来;参见 L. H. Ramshaw, *J. Number Theory* **13** (1981),138~175。]

9.我们有 $\phi^{-1}=/\!/1,1,1,\cdots/\!/$和 $\phi^{-2}=/\!/2,1,1,\cdots/\!/$。设在习题 8 的记号下 $\theta=/\!/\alpha_1,\alpha_2,\cdots/\!/$和 $\theta_k=/\!/a_{k+1},a_{k+2},\cdots/\!/$,以及 $Q_k=q_k+q_{k-1}\theta_{k-2}$。如果 $\alpha_1>2$,则第一次分割是坏的。习题 8 中的三个区间大小分别是$(1-r\theta_{k-1})/Q_k$,θ_{k-1}/Q_k 和 $(1-(r-1)\theta_{k-1})/Q_k$,所以第一个长度对于第二个的比率是$(a_k-r)+\theta_k$。当 $r=$

a_k 和 $a_{k+1}\geqslant 2$ 时，这将小于$\frac{1}{2}$；因此，如果我们不希望有坏的分割，则$\{a_2,a_3,\cdots\}$必须都等于 1。[关于有关的定理，参考 R. L. Graham 和 J. H. van Lint, *Canadian J. Math.* **20** (1968)，1020～1024，以及那里所引的参考文献。]

10. 见 F. M. Liang 在 *Discrete Math.* **28** (1979)，325～326 中漂亮的证明。

11. 如果 K=0，则将是有问题的。如果要求键码如同在程序 L 中那样是非 0 的，则这个变化将是值得的，而且我们也可以用 0 表示空位置。

12. 我们可以把 K 存于 KEY[0]中，用下列诸行代替行 14～19：

```
        STA     TABLE(KEY)      A - S1
        CMPA    TABLE,2(KEY)    A - S1
        JE      3F              A - S1
2H      ENT1    0,2             C - 1 - S2
        LD2     TABLE,1(LINK)   C - 1 - S2
        CMPA    TABLE,2(KEY)    C - 1 - S2
        JNE     2B              C - 1 - S2
3H      J2Z     5F              A - S1
        ENT1    0,2             S2
        JMP     SUCCESS         S2  ▌
```

"节省"的时间是 $C-1-5A+S+4S1$ 个单位，它实际上是一项纯粹的损失，因为 C 很少大于 5。(一个内部循环并不总是优化的!)

13. 如同在算法 C 中那样，设表格的每个表项中有两个可加区分的类型，且有一个附加的二进位 TAG[i]字段。这个解决办法使用循环表，并遵循 Allen Newell 的建议，在每个表的第一个字中有 TAG[i]=1。

A1.[初始化] 置 $i\leftarrow j\leftarrow h(K)+1, Q\leftarrow q(K)$。

A2.[是否有一个表?] 如果 TABLE[i]是空的，则置 TAG[i]←1 而且转到 A8。否则如果 TAG[i]=0，则转到 A7。

A3.[比较] 如果 Q=KEY[i]，则这个算法成功地结束。

A4.[前进到下一个] 如果 LINK[i]$\neq j$，则置 $i\leftarrow$LINK[i]，并转回 A3。

A5.[求空的节点] R 减值一次或多次直到找出一个值，使得 TABLE[R]是空的，如果 $R=0$，则这个算法以溢出结束；否则置 LINK[i]←R。

A6.[准备插入] 置 $i\leftarrow R$，TAG[R]←0，并转到 A8。

A7.[移动一个记录] 重复地置 $i\leftarrow$LINK[i]一次或多次，直到 LINK[i]=j 为止。然后执行步骤 A5，然后置 TABLE[R]←TABLE[i]，$i\leftarrow j$，TAG[j]←1。

A8.[插入新的键码] 标记 TABLE[i]为一个已占用的节点，且 KEY[i]←Q，LINK[i]←j。 ▌

(注意如果 TABLE[i]已被占据的话，仅仅给定 i 的值，就有可能确定对应的完全

的键码 K。我们有 $q(k)=$ KEY(i),而后如果我们置 $i\leftarrow$LINK[i]0 次或多次直到 TAG[i]=1 为止,我们将有 $h(K)=i-1$。)

14. 按照所述的约定,2.3.3-(6)的记号“X⇐AVAIL”现在代表下列操作:“置 X←AVAIL;然后置 X←LINK(X)0 次或多次直到或者 X=Λ(一个溢出错误)或 TAG(X)=0;最后置 AVAIL←LINK(X)。

为了插入一个新的键码 K,置 Q⇐AVAIL,TAG(Q)←1,并把 K 存入这个字中。[或者,如果所有键码都是短的,就省略这个键码,而且在后边以 K 来代替 Q。]然后置 R⇐AVAIL,TAG(R)←1,AUX(R)←Q,LINK(R)←Λ。置 P←$h(K)$。而且

如果 TAG(P)=0,则置 TAG(P)←2,AUX(P)←R;

如果 TAG(P)=1,则置 S⇐AVAIL,CONTENTS(S)←CONTENTS(P),TAG(P)←2,AUX(P)←R,LINK(P)←S;

如果 TAG(P)=2,则置 LINK(R)←AUX(P),AUX(P)←R。

为了恢复一个键码 K:置 P←$h(K)$,而且

如果 TAG(P)≠2,则 K 不存在;

如果 TAG(P)=2,则置 P←AUX(P);然后置 P←LINK(P)0 次或多次直到或者 P=Λ,或 TAG(P)=1,以及或者 AUX(P)=K(如果所有的键码都是短的的话),或 AUX(P)指向包含 K 的一个字(也许间接地通过带有 TAG=2 的字)。

Elcock 开创性的方案[*Comp. J.* **8** (1965),242~243]实际上使用 TAG=2 和 TAG=3 来区分长度为 1 的表(当我们能节省一个字的空间时)和更长的表。这是一个值得的改进,因为我们假设有这样一个很大的散列表,对于它几乎所有的表的长度都为 1。

使用连接而不是分开的拉链,J. S. Vitter 提出了把散列表放在一个很大的链接内存的“顶上”的另外一个方法[*Inf. Proc. Letters* **13** (1981),77~79]。

15. 如果总有一个空的节点,则会使得内部查找循环更快些,因为我们不需要维持一个计数器以确定步骤 L2 已被实施多少次。较短的程序足以补偿这一个浪费的单元。[另一方面,有一种简洁方式省略变量 N 并允许这个表格在算法 L 中变成完全满的,而不显著地减慢这个方法,除非当这个表真正地溢出了:简单地校验 $i<0$ 是否发生两次! 这项技巧不适用于算法 D。]

16. 否:0 总导致 SUCCESS,而无论它已经插入与否,而且 SUCCESS 在不同的时间以不同的 i 值出现。

17. 然后第二个探查将总是对于位置 0 进行。

18. (31)中的代码比(30)多花费大约 $3(A-S1)$单位,它节省了 $4u$ 乘(26)、(27)和(28)、(29)之间的差。对于一次成功的查找,仅当表是 94% 以上满时(31)才是有利的,而且它决不节省多于大约$\dfrac{1}{2}u$ 的时间。对于一次不成功的查找,当这个表是高于 71% 满时,(31)是有利的。

20. 我们要证明

$$\binom{j}{2} \equiv \binom{k}{2} (\text{modulo } 2^m) \quad 和 \quad 1 \leqslant j \leqslant k \leqslant 2^m$$

意味着 $j=k$。我们发现 $j(j-1) \equiv k(k-1)(\text{modulo } 2^{m+1})$ 意味着 $(k-j)(k+j-1) \equiv 0$。如果 $k-j$ 是奇数,则 $k+j-1$ 必须是 2^{m+1} 的一个倍数,但这是不可能的,因为 $2 \leqslant k+j-1 \leqslant 2^{m+1}-2$。因此 $k-j$ 是偶数,所以 $k+j-1$ 是奇数,所以 $k-j$ 是 2^{m+1} 的一个倍数,所以 $k=j$。[反之,如果 M 不是 2 的一个乘幂,则这个探查序列无效。]

探查序列有第二次堆积,而且它使程序 D 的运行时间(如同在(30)中修改的那样)增加大约 $\frac{1}{2}(C-1)-(A-S1)$ 单位,因为 $B \approx \binom{C+1}{3} \Big/ M$ 现在是可以忽略的。在这个表达到大约 60% 满以前,这是一项小的改进。

21. 如果 N 被减少,则算法 D 可能失误,因为它可能达到一种没有可用空间因而无限地循环的状态。另一方面,如果 N 未被减少,则当仍然有空间时,算法 D 可以标志溢出。后一选择的弊病较少,因为重新散列可用来除去被删去的单元。(注意在这种情况下算法 D 将使 N 增值,而且仅当插入一项到一个以前的空位置时才作溢出判断,因为 N 表示非空的位置数。)我们也可以有两个计数器。

22. 假设位置 $j-1, j-2, \cdots, j-k$ 被占用,而且 $j-k-1$ 为空(modulo M)。则被插入之前先探查位置 j 并发现该位置已被占用的那些键码,恰恰是在位置 $j-1$ 到 $j-k$ 之间的那些键码,这些键码的散列地址不处于 $j-1$ 和 $j-k$ 之间;这些有问题的键码以插入的次序出现。算法 R 把第一个这样的键码移动到位置 j 去,并在一个较小的有问题位置的范围内重复这个过程,直到不再剩下有问题的键码为止。

23. J. S. Vitter[*J. Algorithms* **3** (1982), 261～275]设计的接合的拉链的删去方案维持查找时间的分布。

24. $P(P-1)(P-2)P(P-1)P(P-1)/(MP(MP-1)\cdots(MP-6)) = M^{-7}(1-(5-21/M))P^{-1}+O(P^{-2}))$。一般地,一个散列序列 $a_1 \cdots a_N$ 的概率是 $(\prod_{j=0}^{M-1} P^{\underline{b_j}})/(MP)^{\underline{N}} = M^{-N}+O(P^{-1})$,其中 b_j 是等于 j 的 a_i 的个数。

25. 设第 $(N+1)$ 个键码散列到位置 a;P_k 是 M^{-N} 乘使得 k 个位置 $a, a-1, \cdots, a-k+1$(modulo M)被占用和 $a-k$ 个为空的散列序列数。由这个算法的循环对称性,有 $a+1, \cdots, a+t$ 个被占用和 $a+t+1$ 个空的这种序列的数目是 $g(M, N, t+k)$。

26. $\dfrac{9!}{2!\ 2!\ 4!\ 1!} f(3,2) f(3,2) f(5,4) f(2,1) = 2^2 3^5 5^4 7 = 4252500$。

27. 遵循提示

$$s(n,x,y) = \sum_k \binom{n}{k} x(x+k)^k (y-k)^{n-k-1}(y-n) + n\sum_k \binom{n-1}{k-1}(x+k)^k (y-k)^{n-k-1}(y-n)$$

在第一个和中，以 $n-k$ 代替 k，而且应用 Abel 公式；在第二个和中，以 $k+1$ 代替 k，现在

$$g(M,N,k)=\binom{N}{k}(k+1)^{k-1}(M-k-1)^{N-k-1}(M-N-1)$$

而且当 $k=N=M-1$ 时有 $0/0=1$，而且

$$M^N\sum(k+1)P_k=\sum_{k\geqslant 0}\binom{k+2}{2}g(M,N,k)=$$

$$\frac{1}{2}\Big(\sum_{k\geqslant 0}(k+1)g(M,N,k)+\sum_{k\geqslant 0}(k+1)^2g(M,N,k)\Big)$$

第一个和是 $M^N\sum P_k=M^N$，而第二个是 $s(N,1,M-1)=M^N+2NM^{N-1}+3N(N-1)M^{N-2}+\cdots=M^NQ_1(M,N)$。[关于类似于 $s(n,x,y)$的和的进一步研究，见 J. Riordan, *Combinatorial Identities* (New York: Wiley, 1968), 18～23。]

28. 设 $t(n,x,y)=\sum_{k\geqslant 0}\binom{n}{k}(x+k)^{k+2}(y-k)^{n-k-1}(y-n)$；然后如同在习题 27 中那样，我们求得 $t(n,x,y)=xs(n,x,y)+nt(n-1,x+1,y-1)$，$t(N,1,M-1)=M^N(3Q_3(M,N)-2Q_2(M,N))$。于是 $\sum(k+1)^2P_k=M^{-N}\cdot\sum\left(\frac{1}{3}(k+1)^3+\frac{1}{2}(k+1)^2+\frac{1}{6}(k+1)\right)g(M,N,k)=Q_3(M,N)-\frac{2}{3}Q_2(M,N)+\frac{1}{2}Q_1(M,N)+\frac{1}{6}$。减去 $(C'_N)^2$ 给出方差，它近似地等于 $\frac{3}{4}(1-\alpha)^{-4}-\frac{2}{3}(1-\alpha)^{-3}-\frac{1}{12}$。标准差通常大于均值；例如，当 $\alpha=.9$ 时，均值是 50.5 而标准差是 $\frac{1}{2}\sqrt{27333}\approx 82.7$。

29. 设 $M=m+1$，$N=n$；安全的停车序列恰是那样的序列，其中当应用算法 L 于散列序列 $(M-a_1)\cdots(M-a_n)$ 时，位置 0 是空的。因此答案是 $f(m+1,n)=(m+1)^n-n(m+1)^{n-1}$。[这个停车问题是由 A. G. Konheim 和 B. Weiss 开始研究的，见 *SIAM J. Applied Math.* **14** (1966), 1266～1274。]

30. 显然，如果小汽车都停下了，则它们定义了这样一个排列。反之，如果存在 $p_1p_2\cdots p_n$，设 $q_1q_2\cdots q_n$ 是逆排列（$q_i=j$ 当且仅当 $p_j=i$），且设 b_i 是等于 i 的 a_j 的个数。如果我们可以证明 $b_n\leqslant 1$，$b_{n-1}+b_n\leqslant 2$，等等；等价地 $b_1\geqslant 1$，$b_1+b_2\geqslant 2$ 等等，则每一辆小汽车都能停下；但这显然是真的，因为 k 个元素 $a_{q_1},\cdots,a_{q_k}$ 全都 $\leqslant k$。

[设 r_j 是 q_j 的"左影响"，即当且仅当 $q_{j-1}<q_j,\cdots,q_{j-k-1}<q_j$ 且 $j=k$ 或 $q_{j-k}>q_j$ 时 $r_j=k$。在支配一个给定的醒来序列 $a_1\cdots a_n$ 的所有排列 $p_1\cdots p_n$ 当中，"立即停车"算法求得最小者（在字典编辑次序下）。Konheim 和 Weiss 发现，导致一个给定排列 $p_1\cdots p_n$ 的醒来序列数目是 $\prod_{j=1}^{n}\gamma_j$；值得注意的是，取遍所有排列 $q_1\cdots q_n$，这些乘积之和是 $(n+1)^{n-1}$。]

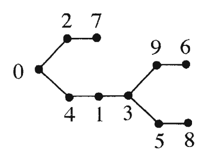
31. 可能有许多有趣的联系，而且下列两个是作者特别喜爱的[也参见 Foata 和

Riordan, *AEquat. Math*. **10** (1974), 10～22]:

a) 在上述答案的记号下,当且仅当($b_1, b_2, \cdots b_n, 0$)是在前根次序下树节点次数的一个正确序列时,计数 $b_1, b_2, \cdots, b_n$ 对应于一个完全的停车序列。(对照 2.3.3－(9)它画出了后根次序。)每株这样的树都对应于{0,…,n}上 $n!/b_1!\cdots b_n!$ 株不同的带标号的自由树,因为我们能让 0 作为根的标号,而且对于 $k=1,2,\cdots,n$,我们可以逐次地由剩下未使用的标号中以 $(b_k+\cdots+b_n)!/b_k!(b_{k+1}+\cdots+b_n)!$ 种方式选择在前根次序下第 k 个节点的儿子的标号,以从左到右递增的次序附加诸标号。而且,每一个这样的计数序列,都对应于 $n!/b_1!\cdots b_n!$ 个醒来序列。

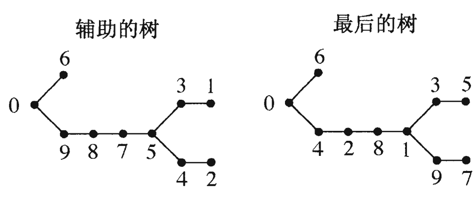
b) Dominique Foata 已经给出下列别致的一一对应关系:设 $a_1\cdots a_n$ 是安全停车序列,它总在空位 j 处停放车 q_j。对于 $1\leqslant j\leqslant n$,当 $a_j=1$ 时从 j 到 0 画一条边,否则从 j 到 q_{a_j-1} 画一条边,就构造出了{0,1,…,n}上的一株带标号的自由树。(把这株树的节点想成是汽车,汽车 j 被连接到另一辆车上,该车恰恰停在妻子 j 醒来处前面的位置上)。

例如,由 Foata 的规则,醒来时间序列 314159265 导致自由树。



反之,假定箭头从根 0 处出发,而且在每一步骤都选择最小的"源",通过拓朴排序可从树得到停车序列。从这个序列,可以重新构造 $a_1\cdots a_n$。

c) 首先通过命节点 k 的父节点是在排列 $q_1\cdots q_n$ 中跟随 k 的 $>k$ 的头一个元素,可构造一株辅助的树;如果没有这样一个元素,命父节点为 0。然后以前根次序,如下述进行,制作辅助树的一个副本并且重新标号新树的非 0 节点:如果在辅助树中当前的节点的标号是 k,把它当前的标号同在它的子树中当前为第 $1+p_k-a_k$ 最小的标号交换,例如



为颠倒这个过程,我们可以通过以前根次序进行把每个节点的标号同在它的子树中当前最大的标号加以交换。

构造(a)和(b)是密切相关的,但构造(c)十分不同。它有这样有趣的性质,即从它们中意的位置开始车的位移之和等于树中反序个数—即标号对 $a>b$ 的个数,其中 a 是 b 的一个祖先。G. Kreweras[*Periodica Math. Hung*. **11** (1980), 309～320]首先发现了停车序列和树的反序之间的这一关系。树的反序同连通图密切相关这一

事实[Mallows 和 Riordan,*Bull. Amer. Math. Soc.* **74** (1968),92～94]现在使得有可能推导出取遍所有停车序列的 $\binom{D(p)}{k}$ 之和,其中 $D(p)=(p_1-a_1)+\cdots+(p_n-a_n)$ 等于在标号顶点 $\{0,1,\cdots,n\}$ 上有 $n+k$ 条边的连通图的总数。[参见 Janson, Łuczak, Knuth 和 Pittel,*Random Structures Alg.* **4** (1993),233～358 的论文中的等式(2.11),(3.5)和(8.13)。]

32. 设

$$s_j = \sum_{k=0}^{j}(b_{k \bmod M}-1)$$

那么,根据 Svante Janson 的观察,我们有 $c_j=\max_{k\geqslant j}(s_k-s_j)$,这个量是有意义的,因为 $\lim_{k\to\infty}s_k=-\infty$。这个解可以通过在 $c_0=0$ 的假定基础上定义 $c_{M-1},c_{M-2},\cdots$ 而找到;然后,如果结果表明 c_0 大于 0,则只需重新定义 $c_{M-1},c_{M-2},\cdots$,直到不再作进一步的改变为止。

33. 各个概率不是独立的,因为条件 $b_0+b_1+\cdots+b_{M-1}=N$ 未考虑在内;这个推导允许 $\sum b_j$ 有任何给定的非负值的概率为非 0。等式(46)不是严格地正确的;例如,它们意味着对于所有的 k,q_k 为正,这同 c_j 决不超过 $N-1$ 的事实矛盾。

Gaston Gonnet 和 Ian Munro[*J. Algorithms* **5** (1984),451－470]已经找到了从导致(51)的论证出发,通过引进对于一个序列 $\langle A_{mn}\rangle$ 的泊松变换,来导出精确结果的一个有趣的方法。我们有 $e^{-mz}\sum_n A_{mn}(mz)^n/n! = \sum_k a_k z^k$,当且仅当 $A_{mn}=\sum_k a_k n^{\underline{k}}/m^k$。

34. (a) 有 $\binom{N}{k}$ 种方式来选择 j 的集合使得 a_j 有一个特定的值,而且有 $(M-1)^{N-k}$ 种方式来赋值给其它的 a。因此

$$P_{Nk}=\binom{N}{k}(M-1)^{N-k}/M^N$$

(b) (50)中 $P_N(z)=B(z)$。

(c)考虑为找出所有键码的探查总数,不计算取图 38 中表列头表内指针的次数,如果使用这样的表的话。长度为 k 的一个表列对总和数贡献了 $\binom{k+1}{2}$;因此

$$C_N = M\sum\binom{k+1}{2}P_{Nk}/N = (M/N)\left(\frac{1}{2}P''_N(1)+P'_N(1)\right)$$

(d) 在情况(i)中长度为 k 的一个表列要求 k 次探查(不计取表头),而在情况(ii)中,它要求 $k+\delta_{k0}$。于是在情况(ii),我们得到 $C'_N=\sum(k+\delta_{k0})P_{Nk}=P'_N(1)+P_N(0)=N/M+(1-1/M)^N\approx\alpha+e^{-\alpha}$,而情况(i)简单地有 $C'_N=N/M=\alpha$。公式 $MC'_N=M-N+NC_N$ 适用于情况(iii),因为 $M-N$ 个散列地址将发现一个空表位置,而 N 将引起去查找某个表列的结尾;这就得到(18)。

35.(i) $\sum\left(1+\frac{1}{2}k-(k+1)^{-1}\right)P_{Nk}=1+N/(2M)-M(1-(1-1/M)^{N+1}/(N+1)\approx 1+\frac{1}{2}\alpha-(1-e^{-\alpha})/\alpha$。(ii) 在(i)的结果中加上$\sum\delta_{k0}P_{Nk}=(1-1/M)^N\approx e^{-\alpha}$。(iii) 假定当一次不成功的查找在长度为 k 的一个表列的第 j 个元素处开始，给定的键码相对于其它 k 个元素有随机次序，所以预期的查找长度是$(j\cdot 1+2+\cdots+(k+1-j)+(k+1-j))/(k+1)$。现在对 j 求和给出 $MC'_N=M-N+M\sum(k^3+9k^2+2k)P_{Nk}/(6k+6)=M-N+M\left(\frac{1}{6}N(N-1)/M^2+\frac{3}{2}N/M-1+(M/(N+1))(1-(1-1/M)^{N+1})\right)$；因此$C'_N\approx\frac{1}{2}\alpha+\frac{1}{6}\alpha^2+(1-e^{-\alpha})/\alpha$。

36.(i) $N/M-N/M^2$。(ii) $\sum(\delta_{k0}+k)^2P_{Nk}=\sum(\delta_{k0}+k^2)P_{Nk}=P_N(0)+P''_N(1)+P'_N(1)$。减去$(C'_N)^2$ 给出答案，即$(M-1)N/M^2+(1-1/M)^N(1-2N/M-(1-1/M)^N)\approx\alpha+e^{-\alpha}(1-2\alpha-e^{-\alpha})\leqslant 1-e^{-1}-e^{-2}=0.4968$。[对于数据结构(iii)，将需要像在习题 37 中那样一个更复杂的分析。]

37.设 S_N 是$(C_N-1)^2$ 的平均值，考虑散列序列的 M^NN 种选择并认为诸键码是同样可能的。则

$$\begin{aligned}M^NNS_N=&\frac{1}{3}\sum\binom{N}{k_1,\cdots,k_M}\left(k_1\left(k_1-\frac{1}{2}\right)(k_1-1)+\cdots+\right.\\&\left.k_M\left(k_M-\frac{1}{2}\right)(k_M-1)\right)=\\&\frac{1}{3}M\sum_k\binom{N}{k}(M-1)^{N-k}k\left(k-\frac{1}{2}\right)(k-1)=\\&\frac{1}{3}MN(N-1)(N-2)\sum_k\binom{N-3}{k-3}(M-1)^{N-k}+\\&\frac{1}{2}MN(N-1)\sum_k\binom{N-2}{k-2}(M-1)^{N-k}=\\&\frac{1}{3}MN(N-1)(N-2)M^{N-3}+\frac{1}{2}MN(N-1)M^{N-2}\end{aligned}$$

方差是 $S_N-((N-1)/2M)^2=(N-1)(N+6M-5)/12M^2\approx\frac{1}{2}\alpha+\frac{1}{12}\alpha^2$。

关于在这里计算的总方差和两个其它的方差思想之间有趣的关系，参见 *CMath*. §8.5。这两个方差是平均探查次数(取遍所有存在的表项)的方差(取遍随机散列表)，和探查次数(取遍所有存在的项)的方差的平均(取遍随机散列表)。总方差总是其它两者之和。而在这个情况下探查的平均数的方差是$(M-1)(N-1)/(2M^2N)$。

38.由等式 6.2.2-(5)和 6.2.2-(6)，在不成功的情况下平均探查次数为

$\sum P_{Nk}(2H_{k+1}-2+\delta_{k0})$，在成功的情况下为$(M/N)\sum P_{Nk}k(2(1+1/k)H_k-3)$。这些和分别等于$2f(N)+2M(1-(1-1/M)^{N+1}/(N+1)+(1-1/M)^N-2$和$2(M/N)f(N)+2f(N-1)+2M(1-(1-1/M)^N)/N-3$，其中$f(N)=\sum P_{Nk}H_k$。习题5.2.1-40告诉我们当$N=\alpha M$，$M\to\infty$时$f(N)=\ln\alpha+\gamma+E_1(\alpha)+O(M^{-1})$。

[树散列首先是由 P.F.Windley 提出的，参见*Comp.J.* **3** (1960)，84～88。在上一段中的分析表明树散列不比简单的拉链好到足以证明额外的链接字段的正确性；表列无论如何总是短的；而且当 M 很小时，它不比纯粹树查找好到足以认为散列时间合理。]

39.（对算法 C 的分析的这一方法是由 J.S.Vitter 提出的。）当 $k\geqslant 2$ 时，我们有$c_{N+1}(k)=(M-k)c_N(k)+(k-1)c_N(k-1)$，而且$\sum kc_N(k)=NM^N$。因此

$$S_{N+1}=\sum_{k\geqslant 2}\binom{k}{2}c_{N+1}(k)=$$
$$\sum_{k\geqslant 2}\binom{k}{2}((M-k)c_N(k)+(k-1)c_N(k-1))=$$
$$\sum_{k\geqslant 1}\left((M+2)\binom{k}{2}+k\right)c_N(k)=$$
$$(M+2)S_N+NM^N$$

因此

$$S_N=(N-1)M^{N-1}+(N-2)M^{N-2}(M+2)+\cdots+M(M+2)^{N-2}=$$
$$\frac{1}{4}(M(M+2)^N-M^{N+1}-2NM^N)。$$

考虑在不成功的查找中探查的总数，对 $h(K)$ 的所有 M 个值求和；长度为 k 的每个表列对总和贡献$k+\delta_{k0}+\binom{k}{2}$，因此$M^{N+1}C'_N=M^{N+1}+S_N$。

40.像习题 39 中的 S_N 那样定义 U_N，但以$\binom{k+1}{3}$来代替$\binom{k}{2}$。我们求出$U_{N+1}=(M+3)U_N+S_N+NM^N$，因此$U_N=\frac{1}{36}(M^N(M-6N)-9M(M+2)^N+8M(M+3)^N)$。方差为$2U_N/M^{N+1}+C'_N-(C'_N)^2$，对于$N=\alpha M$，$M\to\infty$，它趋于

$$\frac{35}{144}-\frac{1}{12}\alpha-\frac{1}{4}\alpha^2+\left(\frac{1}{4}\alpha-\frac{5}{8}\right)e^{2\alpha}+\frac{4}{9}e^{3\alpha}-\frac{1}{16}e^{4\alpha}$$

当 $\alpha=1$ 时，这大约是 4.50，所以标准差大约以 2.12 为界。

41.设 V_N 是在表格的“高”端被占用单元的块区的平均长度。这个块区的长度为 k 的概率是$A_{Nk}(M-1-k)^{\underline{N-k}}/M^N$，其中 A_{Nk} 是具如下性质的散列序列(35)的个数，它使得在算法 C 作用下，前 $N-k$ 个和最后 k 个单元为已占用的，并且使得子序列 $1,2,\cdots,N-k$ 以递增次序出现。因此

$$M^NV_N=\Sigma_kA_{Nk}(M-1-k)^{\underline{N-k}}=M^{N+1}-\Sigma_k(M-k)A_{Nk}(M-1-k)^{\underline{N-k}}=$$

$$M^{N+1} - (M-N)\Sigma_k A_{Nk}(M-k)^{\underline{N-k}} = M^{N+1} - (M-N)(M+1)^N$$

现在 $T_N = (N/M)(1+V_N - T_0 - \cdots - T_{N-1})$，因为 $T_0 + \cdots + T_{N-1}$是在此之前 R 减值的平均次数，且 N/M 是在当前的步骤减值的概率。这个递推式的解是 $T_N = (N/M)(1+1/M)^N$。（这样一个简单的公式应该有一个更简单的证明！）

42. $S1_N$是以 $A=0$ 被插入的项数除以 N。

43. 设 $N=\alpha M'$和$M=\beta M'$，并设 $e^{-\lambda}+\lambda=1/\beta$，$\rho=\alpha/\beta$。则若 $\rho\leqslant\lambda$，就有$C_N\approx 1+\frac{1}{2}\rho$ 和$C'_N\approx\rho+e^{-\rho}$；若 $\rho\geqslant\lambda$，则 $C_N\approx\frac{1}{8\rho}(e^{2\rho-2\lambda}-1-2\rho+2\lambda)(3-2/\beta+2\lambda)+\frac{1}{4}(\rho+2\lambda-\lambda^2/\rho)$和 $C'_N\approx 1/\beta+\frac{1}{4}(e^{2\rho-2\lambda}-1)(3-2/\beta+2\lambda)-\frac{1}{2}(\rho-\lambda)$。对于 $\alpha=1$，当$\beta\approx.853$时，我们得到最小的 $C_N\approx1.69$；当 $\beta\approx.782$ 时出现最小的 $C'_N\approx 1.79$。对于一个宽范围的 α，设置 $\beta=.86$ 给出近乎最优的查找性能。所以值得把头一个冲突放在不同散列地址发生冲突的一个区域中，尽管较小范围的散列地址将引起更多的冲突出现，这些结果是由 Jeffrey S. Vitter, *JACM* **30** (1983)，231～258 给出的。

44.（下列硬算的方法是作者在 1972 年找到的解；由 M.S. Paterson 给出的一个更漂亮的解由 Greene 和 Knuth 在*Mathematics for the Analysis of Algorithms* (Birkhäuser Boston,1980) §3.4 中作了说明。Paterson 还发现了简化本节中若干其它分析的重要方法。）

从左到右，以 1 到 m 给阵列位置编号。考虑同等可能地具有 k 个"p 步骤"和 $n-k$ 个"q 步骤"的所有$\binom{n}{k}$个操作序列的集合，设 $g(m,n+1,k,r)$是$\binom{n}{k}$乘上前 $r-1$ 个位置变成已占用的和第 r 个保持为空的概率。于是 $g(m,l,k,r)$是$(m-1)^{-(l-1-k)}$乘以下述的概率取遍所有配置

$$1\leqslant a_1<\cdots<a_k<l;\ (c_1,\cdots,c_{l-1-k}),\ 2\leqslant c_i\leqslant m$$

之和：所述概率是当第 a_j 个操作是一个 p 步骤，而剩下的 $l-1-k$ 个操作是分别以选择位置$c_1,\cdots,c_{l-1-k}$开始的诸q 步骤时，第一个空位置是 r 的概率。通过附加进一步的条件，即给定 $1\leqslant b_1<\cdots<b_k<r$ 第a_j 个操作占用位置b_j，而后对所有的配置求和，我们得到递推式

$$g(m,l,k+1,r) = \sum_{\substack{a<l\\ b<r\\ 1\leqslant b\leqslant a}} \frac{(l-b-1)!}{(l-r)!}\frac{(m-r)!}{(m-b)!}(m-l+1)g(m,a,k,b)$$

$$g(m,l,0,r) = \frac{(l-1)!}{(l-r)!}\frac{(m-r)!}{m!}(m-l+1)\left(P_l + [r\neq 1]\frac{m}{l-1}(1-P_l)\right)$$

其中 $P_l=(m/(m-1))^{l-1}$。令 $G(m,l,k)=\sum_{r=1}^{l}(m+1-r)g(m,l,k,r)$，由此得出，

$$G(m,l,k+1) = \frac{m-l+1}{m-l+2}\sum_{a=1}^{l-1}G(m,a,k);$$

$$G(m,l,0)=\frac{m-l+1}{m-l+2}(m+P_l)$$

对于所述问题的答案是 $m-\sum_{k=0}^{n}p^kq^{n-k}G(m,n+1,k)$，它(在实施了某些技巧之后)等于 $m-((m-n)/(m-n+1))(Q_n+mR+pSR)$，其中

$$Q_j=P_{j+1}q^j,$$

$$R=\left(1-\frac{p}{m+1}\right)\left(1-\frac{p}{m}\right)\cdots\left(1-\frac{p}{m-n+2}\right)=\prod_{j=0}^{n-1}\left(1-\frac{p}{m+1-j}\right),$$

$$S=\frac{\left(1-\frac{1}{m+1}\right)Q_0}{\left(1-\frac{p}{m+1}\right)}+\frac{\left(1-\frac{1}{m}\right)Q_1}{\left(1-\frac{p}{m+1}\right)\left(1-\frac{p}{m}\right)}+\cdots+\frac{1-\left(\frac{1}{m-n+2}\right)Q_{n-1}}{R}=$$

$$\sum_{k=0}^{n-1}\frac{(1-1/(m+1-k))Q_k}{\prod_{j=0}^{k}(1-p/(m+1-j))}$$

当 $p=1/m$ 时，对于所有的 j，$Q_j=1$。设 $w=m+1$，$n=\alpha w$，$w\to\infty$，我们求得 $\ln R=-(H_w-H_{w(1-\alpha)})p+O(p^2)$；因此 $R=1+w^{-1}\ln(1-\alpha)+O(w^{-2})$；而且类似地 $S=\alpha w+O(1)$。于是答案是 $(1-\alpha)^{-1}-1-\alpha-\ln(1-\alpha)+O(w^{-1})$。

注意：较简单的问题"以概率 p 占用最左边，否则占用任何随机选定的空位置"，通过在上边的公式中取 $P_j=1$ 即可获得解决，而且答案是 $m-(m+1)(m-n)R/(m-n+1)$。为了得到有第二次堆积的随机探查的 C_N'，置 $n=N$，$m=M$，并在上边的答案中加 1。

45. 是的。见 L. Guibas，*JACM* **25** (1978)，544～555。

46. 对于所有的 x 和非负整数 n 通过规则

$$\sum_k\binom{x+k}{k}\left[\left[\begin{matrix}n\\k\end{matrix}\right]\right]=(x+n+1)^n$$

对于 $k\geqslant0$ 定义数 $\left[\left[\begin{matrix}n\\k\end{matrix}\right]\right]$。置 $x=-1,-2,\cdots n-1$ 意味着

$$\left[\left[\begin{matrix}n\\k\end{matrix}\right]\right]=\sum_j\binom{k}{j}(-1)^j(n-j)^n\text{对于 }0\leqslant k\leqslant n;$$

然后置 $x=0$ 意味着对于所有 $k>n$，我们可以取 $\left[\left[\begin{matrix}n\\k\end{matrix}\right]\right]=0$，所以定义方程的两边是在 $n+1$ 个点上相等的 x 的 n 次多项式，由此得出数 $\left[\left[\begin{matrix}n\\k\end{matrix}\right]\right]$ 有所述的性质。

设 $f(N,r)$ 是保持头 r 个为已占用的和下一个为空的散列序列 $a_1\cdots a_N$ 的数目。有 $\binom{M-r-1}{N-r}$ 种占用单元的型式，而且每一种型式出现的次数就等于序列 $a_1'\cdots a_N'$，$1\leqslant a_i'\leqslant N$ 的数目，这种序列包含数 $r+1,r+2,\cdots,N$ 的每一个至少一

次。按容斥原理有$\left[\left[\begin{matrix}N\\N-r\end{matrix}\right]\right]$个这样的序列；因此

$$f(N,r)=\binom{M-r-1}{N-r}\left[\left[\begin{matrix}N\\N-r\end{matrix}\right]\right]$$

现在$C'_N=1+M^{-N-1}\sum_{r=0}^{N}f(N,r)\left(\sum_{a=0}^{r-1}r+\sum_{a=r+1}^{M-1}\frac{N-r}{M-r-1}(r+1)\right)=$

$$1+M^{-N-1}\sum_{r=0}^{N}f(N,r)(N+(N-1)r)$$

设 $S_n(x)=\Sigma_k k\binom{x+k}{k}\left[\left[\begin{matrix}n\\k\end{matrix}\right]\right]$；我们有

$$(x+1)^{-1}S_n(x)+\sum_k\binom{x+k}{k}\left[\left[\begin{matrix}n\\k\end{matrix}\right]\right]=\sum_k\binom{x+1+k}{k}\left[\left[\begin{matrix}n\\k\end{matrix}\right]\right];$$

因此，$S_n(x)=(x+1)((x+n+2)^n-(x+n+1)^n)$。由此得出 $C'_N=N(1+1/M)-(N-1)(1-N/M)(1+1/M)^N\approx N(1-(1-\alpha)e^{\alpha})$；而且 $C_N=(N-1)((1+1/M)/2+(1+1/M)^N)+(3M^2+6M+2)((1+1/M)^N-1)/N-(3M+2)(1+1/M)^N$，当 $N=M-1$时，它是$(e-2.5)M+O(1)$。

关于数$\left[\left[\begin{matrix}n\\k\end{matrix}\right]\right]$进一步的性质，参见 John Riordan，*Combinatorial Identities*（New York Wiley，1968），228～229。

47. 算法 L 的分析几乎可逐字适用！任何带有循环对称性的探查序列，而且它仅仅勘察同从前考察过的那些位置相邻的位置，都将有相同的特性。

48. $C'_N=1+p+p^2+\cdots$，其中 $p=N/M$ 是一个随机位置被填满的概率；因此$C'_N=M/(M-N)$，而且 $C_N=N^{-1}\sum_{k=0}^{N-1}C'_k=N^{-1}M(H_M-H_{M-N})$。这些值近似地等于一致探查时的那些值，但稍微高些，因为在相同的位置有多次探查的机会。其实对于 $4=N<M\leqslant16$，线性探查是更好的！

实际上，我们将不使用无穷多个散列函数，某些其它的方案，像线性探查，将最终地被用作最后的手段。这个方法比正文中描述的那些方法低劣，但是它有着历史上的重要性，因为它提示了导致算法 D 的 Morris 方法。见*CACM* **6**（1963），101，其中 M.D.McIlroy 把这个思想归功于 V.A.Vyssotsky；同样的技术也由 A.W.Holt 早在 1956 年就发现了，他成功地把它使用于 UNIVAC 的 CPX 系统中。

49. $C'_N-1=\sum_{k>b}(k-b)P_{Nk}\approx\sum_{k>b}(k-b)\mathrm{e}^{-\alpha b}(\alpha b)^k/k!\ =\alpha bt_b(\alpha)$。注：一般说来，如果 $P(z)=P_0+P_1z+\cdots$是任何概率的生成函数，则

$$\sum_{b\geqslant0}\Big(\sum_{k>b}(k-b)P_k\Big)z^b=\frac{P'(1)}{1-z}+\frac{z(P(z)-1)}{(1-z)^2}$$

而且，
$$C_N-1=\frac{M}{N}\sum_{k>b}\binom{k-b+1}{2}P_{Nk}=$$

$$\frac{M}{2N}\sum_{k>b}(k(k-1)-2k(b-1)+b(b-1))P_{Nk}\approx$$

$$\frac{1}{2}e^{-b\alpha}(b\alpha)^b b!^{-1}(b+b\alpha-2b+2+$$

$$(b\alpha^2-2\alpha(b-1)+b-1)R(\alpha,b))$$

[对于具有拉链的成功查找的分析,首先是由 W.P.Heising 于 1957 年进行的。(57)和(58)中的简单表达式是由 J.A.Van der Pool 于 1971 年建立的;他也考虑了怎样把一个表示存储空间和存取次数的组合代价的函数极小化。因为 $\sum_{k>b}(k-b)^2P_{Nk}=(2N/M)(C_N-1)-(C'_N-1)$,我们可以确定 C'_N 和每个桶溢出次数的方差。溢出总数的方差可以通过把 M 乘以在一个桶中的方差来逼近,但这实际上太高了,因为总的记录数已被限制为 N。真正的方差可以像在习题 37 中那样来求得。也请参见3.3.1C节中的 χ 平方测试的推导。]

50.而且其次 $Q_0(M,N-1)=(M/N)(Q_0(M,N)-1)$。一般地,$rQ_r(M,N)=MQ_{r-2}(M,N)-(M-N-r)Q_{r-1}(M,N)=M(Q_{r-1}(M,N+1)-Q_{r-1}(M,N))$;$Q_r(M,N-1)=(M/N)(Q_r(M,N)-Q_{r-1}(M,N))$。

51.$R(\alpha,n)=\alpha^{-1}(n!\ e^{\alpha n}(\alpha n)^{-n}-Q_0(\alpha n,n))$。

52.参见等式 1.2.11.3-(9)和习题 3.1-14。

53.由等式 1.2.11.3-(8),$\alpha(\alpha n)^nR(\alpha,n)=e^{\alpha n}\gamma(n+1,\alpha n)$;因此由提示的习题可知 $R(\alpha,n)=(1-\alpha)^{-1}-(1-\alpha)^{-3}n^{-1}+O(n^{-2})$。[这个渐近公式可以更直接地通过(43)的方法得到,如果注意 $R(\alpha,n)$ 中 α^k 的系数是

$$1-\binom{k+2}{2}n^{-1}+O(k^4n^{-2})$$

的话。事实上,由等式 1.2.9-(28),α^k 的系数是

$$\sum_{r\geqslant 0}(-1)^r n^{-r}\left\{{r+k+1 \atop k+1}\right\}。]$$

54.使用提示和等式 1.2.6-(53)和 1.2.6-(49)一起,我们有

$$\sum_{b\geqslant 1}t_b(\alpha)=\sum_{m\geqslant 1}\frac{\alpha^m}{(m+1)(m)m!}\sum_k\binom{m}{k}(-1)^{m-k}k^{m+1}=\sum_{m\geqslant 1}\alpha^m/2$$

由于 $(n+1)!\ t_n(\alpha)=e^{-n\alpha}(\alpha n)^nF(2;n+2;\alpha n)$,从 Kummer 著名的超几何恒等式 $e^{-z}F(a;b;z)=F(b-a;b;-z)$ 可得出提示;见 *Crelle* **15**(1836),39~83,127~172,等式 26.4。

55.如果 $B(z)C(z)=\sum s_iz^i$,则我们有 $c_0=s_0+\cdots+s_b$,$c_1=s_{b+1}$,$c_2=s_{b+2}$,…;因此 $B(z)C(z)=z^bC(z)+Q(z)$。现在 $P(z)=z^b$ 有 $b-1$ 个根 q;且 $|q_j|<1$,如同对 $e^{\alpha(q_j-1)}=\omega^{-j}q_j$,$\omega=e^{2\pi i/b}$ 的解那样确定。为了求解 $e^{\alpha(q-1)}=\omega^{-1}q$,设 $t=\alpha q$ 和 $z=\alpha\omega e^{-\alpha}$,使得 $t=ze^t$。由拉格朗日公式我们得到

$$\frac{1}{1-q}=1+\sum_{r\geqslant 0}r\sum_{n\geqslant r}\frac{n^{n-r-1}\omega^n\alpha^{n-\gamma}e^{-n\alpha}}{(n-r)!}=$$

$$1+\sum_{r\geqslant 1} r \sum_{m\geqslant 0} \frac{\alpha^m}{m!}(-1)^m \sum_{n\geqslant r}\binom{m}{n-r}(-1)^{n-r}\omega^n n^{m-1}$$

由阿贝尔极限定理，从单位圆内命 $|\omega|\to 1$，这可以重新安排成

$$\frac{1-\alpha\omega}{1-\omega}+\sum_{m\geqslant 2}\frac{\alpha^m}{m!}(-1)^m\sum_{n\geqslant 0}\binom{m-2}{n}(-1)^n\omega^{n+1}(n+1)^{m-1}$$

现在以 ω^j 代替 ω，而且对于 $1\leqslant j<b$ 求和，产生

$$\frac{b-1}{2}+\alpha\frac{b-1}{2}+\sum_{m\geqslant 2}\alpha^m\left(-\frac{1}{2}+\frac{(-1)^m}{m!}b\sum_{n\geqslant 1}\binom{m-2}{nb-1}(-1)^{nb-1}(nb)^{m-1}\right)$$

在使用习题 54 的提示稍做修改之后，即得所希望的结果。

这项分析，可应用于广泛的问题中，它是由 N. T. J. Bailey，*J. Roy. Stat. Soc.* **B16** (1954)，80～87；M. Tainiter，*JACM* **10** (1963)，307～315；A. G. Konheim 和 B. Meister，*JACM* **19** (1972)，92～108 开始进行的。

56. 参见 Blake 和 Konheim，*JACM* **24** (1977)，591～606。Alfredo Viola 和 Patricio Poblete[*Algorithmica* **21** (1998)，37～71]已经证明

$$C_{Mb}=1+\frac{M-1}{2Mb}+\frac{1}{b}\sum_{j\geqslant 2}\binom{bm-1}{j}m^{-j}\sum_{k\geqslant 1}\binom{j-2}{bk-1}(-1)^{j+bk-1}k^{j-1}=$$

$$\sqrt{\frac{\pi M}{8b}}+\frac{1}{3b}+\frac{1}{b}\sum_{j=1}^{b-1}\frac{1}{(1-T(e^{2\pi ij/b-1}))}+\frac{1}{24}\sqrt{\frac{\pi}{2b^3M}}+O(b^{-2}M^{-1})$$

其中 T 是等式 2.3.4.4-(30)的树函数。

58. 0 1 2 3 4 和 0 2 4 1 3，加 1 1 1 1 1 mod 5 的附加的移位，每个有 $\frac{1}{10}$ 的概率。类似地，对于 $M=6$，我们需要 30 个排列，而且存在一个以 $\frac{1}{20}\times$0 1 2 3 4 5，$\frac{1}{60}\times$0 1 3 2 5 4，$\frac{1}{60}\times$0 2 4 3 1 5，$\frac{1}{20}\times$0 2 3 4 5 1，$\frac{1}{30}\times$0 3 4 1 2 5 开始的解。对于 $M=7$，我们需要 49 个排列，而且通过 $\frac{1}{35}\times$0 1 2 3 4 5 6，$\frac{2}{105}\times$0 1 5 3 2 4 6，$\frac{1}{35}\times$0 2 4 3 5 1 6，$\frac{2}{105}\times$0 2 6 3 1 4 5，$\frac{1}{35}\times$0 3 6 1 4 2 5，$\frac{1}{105}\times$0 3 2 6 4 1 5，$\frac{1}{105}\times$0 3 1 5 4 2 6 生成的一个解。

59. 任何排列不能有大于

$$1\Big/\binom{M}{\lfloor M/2\rfloor}$$

的概率，所以必然至少有

$$\binom{M}{\lfloor M/2\rfloor}=\exp(M\ln 2+O(\log M))$$

个排列有非 0 的概率。

60. Ajtai，Komlós 和 Szemerédi 已经得到预备性的结果，*Information Processing*

Letters **7** (1978),270～273。

62.参见在*AMM* **81** (1974),323～343 中的讨论,其中对于 $M\leqslant 9$ 显示了最好的循环散列序列。

63.由习题 3.3.2-8,MH_M;标准差是$\approx\pi M/\sqrt{6}$。

64.移动的平均数等于$\frac{1}{2}(N-1)/M+\frac{2}{3}(N-1)(N-2)/M^2+\frac{3}{4}(N-1)(N-2)(N-3)/M^3+\cdots\approx\frac{1}{1-\alpha}-\frac{1}{\alpha}\ln\frac{1}{1-\alpha}$。[在*Comp. J.* **17** (1974),139～140 上解决了一个等阶的问题。]

65.诸键码可以被存储在一个顺序分配的独立的表格中(假定,如果有删去的话,它是按 LIFO 方式进行的)。散列表的项指向这个"名字表";例如 TABLE[i]可以有形式

			L_i	KEY[i]

其中 L_i 是存储在单元 KEY[i],KEY[i]$+1,\cdots$处的键码中的字数。

散列表剩下的项可以以若干方式中的任何一种方式来使用:(a) 作为算法 C 中的一个链接。(b) 作为同键码相关联的信息的一部分,或者(c) 作为"第二个散列码"。后一个思想,是由 Robert Morris 提出的,有时可加速一个查找[对于某函数 $h_2(K)$,仅当 $h_2(K)$匹配它的第二个散列码时,我们才仔细地考察 KEY[i]中的键码]。

66.是;而且诸记录的安排是惟一的;每一次不成功的查找的平均探查数被减少成 C_{N-1},尽管当插入第 N 项时它保留 C'_N。这种重要的技术称为有序散列。见 *Comp. J* . **17** (1974),135～142。D.E.Knuth, *Literate Programming* (1992), 144～149,216～217。

67.(a) 如果在(44)中 $c_j=0$,假定 $j-1>\cdots>0>M-1>\cdots>j$,通过把诸 a 排成非递增的"循环次序"得到一个最优的安排。(b) 在步骤 L2 和 L3 之间,交换手中的记录与 TABLE[i],如果后者比前者更接近于家的话。[被 Celis,Larson 和 Munro 在*FOCS* **26** (1985),281～288 中称做"Robin Hood 散列"的这个算法,等价于有序散列的一个变形。](c) 令 $h(m,n,d)$是使 $c_0\leqslant d$ 的散列序列的个数。可以证明[*Comp. J.* **17** (1974),141。]$(h(m,n,d)-h(m,n,d-1))M$ 是在所有 M^N 个散列序列当中位移 $d>0$ 出现的总数,而且我们可以写 $\sum_{k=0}^{d}\binom{n}{k}(m+d-k)^{n-k}(k-d)^k$。现在使用习题 28 和 50 的方法经精心计算证明 $\sum d_j^2$ 的平均值是,当 $N=\alpha M$ 时

$$M^{1-N}\sum_{d=1}^{N}d^2(h(M,N,d)-h(M,N,d-1))=$$
$$\frac{M^2}{2}+\frac{2M}{3}+\frac{N}{6}+\frac{N^2}{6M}-\frac{N}{6M}-M\left(\frac{M}{2}-\frac{N}{2}+\frac{2}{3}\right)Q_0(M,N)=$$

$$M\left(\frac{1}{2(1-\alpha)^2}-\frac{7}{6(1-\alpha)}+\frac{2}{3}+\frac{\alpha}{6}+\frac{\alpha^2}{6}\right)+O(1)$$

无需修改(参见习题 28),$\mathrm{E}\sum d_j^2$ 等于

$$\frac{M}{3}(Q_2(M,N)-Q_1(M,N))-\frac{M}{2}(Q_0(M,N)-1)+\frac{N}{6}=$$

$$M\left(\frac{1}{3(1-\alpha)^3}-\frac{1}{3(1-\alpha)^2}-\frac{1}{2(1-\alpha)}+\frac{1}{2}+\frac{\alpha}{6}\right)+O(1)$$

如果所有记录都有近似地相同的位移 d,而且如果成功的查找比不成功的查找更普遍得多,则在位置 $h'=h(K)+d$ 处开始是有利的。然后探查 $h'-1,h'+1,h'-2$,等等。P. V. Poblete, A. Viola 和 J. I. Munro 已经证明[*Raudom Structures and Algorithms* **10** (1997),221～255]通过使用一个称做“后进先服务”的散列,可以使 $\sum d_j^2$几乎同在 Robin Hood 方法中一样小。在“后进先服务”散列中,每个新插入的键码被放置在它的家的位置当中,所有其它键码移开一步直到找到一个空位为止。Robin Hood 和后进先服务技术既适用于线性探查,也适用于双倍散列,但是相对于双倍探查来说,探查的减少不能补偿对于每次探查所增加的时间,除非表是极其满的。(参见 Poblete 和 Munro, *J. Algorithms* **10** (1989),228～248。)

68. 使用在习题 31 的答案中提到的停车问题和连通图问题之间的联系,可以证明$(d_1+\cdots+d_N)^2$ 的平均值等于

$$\frac{N}{12}((M-N)^3+(N+3)(M-N)^2+(8N+1)(M-N)+5N^2+4N-1-$$
$$((M-N)^3+4(M-N)^2+((6N+3)(M-N)+8N)Q_0(M,N-1))$$

为了得到在一次成功的查找中探查的平均次数的方差,以 N^2 除和减去$\frac{1}{4}(Q_0(M,N-1)-1)^2$;这近似于$\frac{1}{12}((1+2\alpha)/(1-\alpha)^4-1)N+O(N^{-2})$。(参见 P. Flajolet, P. V. Poblete 以及 A. Viola, *Algorithmica* **22** (1998),490～515;D. E. Knuth, *Algorithmica* **22** (1998),561～568。这里所计算的方差应同总方差加以区别,总方差是 $\mathrm{E}\sum d_j^2/N-\frac{1}{4}(Q_0(M,N-1)-1)^2$;参见习题 37 和 67 的答案。)

69. 设 $q_k=p_k+p_{k+1}+\cdots$,于是不等式 $q_k\geqslant\max(0,1-(k-1)(M-n)/M)$给出对于 $C'_N=\sum_{k\geqslant 1}q_k$ 的一个下限。

70. 由 Lueker 和 Molodowitch[*Combinatorica* **13** (1993),83～96]给出的一个特别简单的证明确立了一个类似的结果但在 O 的上限中有一个额外的因子$(\log M)^2$:通过使用更准确的概率估计,以类似的方式可得出所述结果。A. Siegel 和 J. P. Schmidt 已经证明,事实上,在双倍散列中探查的预期次数是 $1/(1-\alpha)+O(1/M)$,其中 $\alpha=N/M$。[*Computer Science Tech. Report* **687** (New York: Courant 研究所,1995)。]

72. [*J. Comp. Syst. Sci.* **18** (1979),143～154。](a) 给定键码 $K_1,\cdots,K_N$ 和 K,

K_j 和 K 在同一表列中的概率是,如果 $K \neq K_j$,则它 $\leqslant 1/M$。因此预期的表列大小 $\leqslant 1+(N-1)/M$。

(b) 假设有 Q 个可能的字符;则对于每个 h_j 有 M^Q 个可能的选择。随机选择每个 h_j 等价于从 M^{Ql} 行和 Q^l 列的矩阵 H 选择一随机行,并且在列 $x_1\cdots x_l$ 中有表项 $h(x_1\cdots x_l)=(h_1(x_1)+\cdots h_l(x_l)) \bmod M$。在列 $K=x_1\cdots x_l$ 和 $K'=x'_1\cdots x'_l$ 且对于某个 j, $x_j \neq x'_j$ 中,我们有 $h(K)=(s+h_j(x_j)) \bmod M$ 和 $h(K')=(s'+h_j(x'_j)) \bmod M$,其中 $s=\sum_{i\neq j}h_i(x_i)$ 和 $s'=\sum_{i\neq j}h_i(x'_i)$ 同 h_j 无关。$h_j(x_j)-h_j(x'_j)$ 的值在 modulo M 之下一致分布;因此不管 s 和 s' 的值如何,我们以 $1/M$ 的概率有 $h(K)=h(K')$。

(c) 是的,加上一个常数到 $h_j(x_j)$ 使 $h(x)$ 改变一个常数 modulo M。

73.(i) 当每个键码被当作是二进位的而非字符的一个序列时[它早在 1970 年就由 Alfred L. Zobrist 发明了。他原来的技术报告已在*ICCA J.* **13** (1990),69～73 上重新发表],这是习题 72(c)的特殊情况。(ii) (b)的证明表明,当 $x_j \neq x'_j$ 时,只要证明 $h_j(x_j)-h_j(x'_j)$ 是一致的 modulo M 即可。而且事实上,对于任何给定的 y 和 y', $h_j(x_j)=y$ 和 $h_j(x'_j)=y'$ 的概率是 $1/M^2$,因为对于任何给定的 (y,y') modulo 素数 M,同余式 $a_jx_j+b_j\equiv y$ 和 $a_jx'_j+b_j\equiv y'$ 有惟一的解 (a_j,b_j)。

当 M 不是素数且 p 是一个 $>M$ 的素数时,如果我们令 $h_j(x_j)=((a_jx_j+b_j) \bmod p) \bmod M$,其中 a_j 和 b_j 是随机地选择 mod p 的。在这种情况下这个族系不是十分广泛的,但对于实用目的来说接近于足够用了。不同的键码冲突的概率至多是 $1/M+r(M-r)/Mp^2 \leqslant 1/M+M/4p^2$,其中 $r=p \bmod M$。

74.一般地说这个命题为假。例如,假设 $M=N=n^2$,并考虑具有 $\binom{N}{n}$ 行以及在不同的列中各以不同方式放上 n 个 0 这样一个矩阵 H。在每行中非 0 元素从左到右为 $1,2,\cdots,N-1$。这个矩阵是通用的因为在每对的列中有 $\binom{N-2}{n-2}=\binom{N}{n}\frac{n}{N}\frac{n-1}{N-1}<\binom{N}{n}\left(\frac{n}{N}\right)^2=R/M$ 个匹配。但在每行的 0 的个数是 $\sqrt{N}\neq O(1)+O(N/M)$。

注意:本习题指出,当插入一个新键码时,预期的表大小十分不同于预期的冲突数。考虑命 $h(x_1\cdots x_l)=h_1(x_1)$,其中 h_1 随机选择。这个散列函数族使每个表的预期大小为 N/M;但它肯定不是通用的,因为有相同的头字符 x_1 的 N 个键码的集合将导致大小为 N 的一个表和所有其它空的表。预期的冲突数将是 $N(N-1)/2$,但对于通用的散列族,这个数至多是 $N(N-1)/2M$,不论键码的集合如何。

另一方面,我们可以证明在一个通用的族中,预期的每个表的大小是 $O(1)+O(N/\sqrt{M})$。假设在行 h 中有 z_h 个 0。于是这行至少包含 $\binom{z_h}{2}$ 对相等的元素。当

每个 z_h 等于 z 时，在 $\sum_{h=1}^{R}\binom{z_h}{2}\leqslant\binom{N}{2}R/M$ 的条件下 $\sum_{h=1}^{R}x_h$ 的极大值出现，其中 $\binom{z}{2}=\binom{N}{2}/M$ 即

$$z=\frac{1}{2}+\sqrt{\frac{1}{4}+\frac{N(N-1)}{M}}<1+\frac{\sqrt{N(N-1)}}{M}$$

75.(a) 显然正确，即使 $h_z,\cdots,h_l$ 恒等于 0。(b) 由 72(b)的答案，正确。(c) 正确。如果 K,K' 和 K'' 全都在某个字符位置中不同，则结果显然。否则，说 $x_j=x_j'\neq x_j''$ 和 $x_k\neq x_k'=x_k''$。则量 $h_j(x_j)+h_k(x_k)$，$h_j(x_j)+h_k(x_k')$ 和 $h_j(x_j'')+h_k(x_k')$ 是彼此无关、一致分布的，而且也和这些键码的其它 $l-2$ 个字符无关。(d) 假的。考虑有 1 个二进位字符的情况 $M=l=2$。于是所有四个键码以 1/4 的概率散列到相同的位置。

76.使用 $h(K)=(h_0(l)+h_1(x_1)+\cdots+h_l(x_l))\bmod M$，其中每个 h_j 如同在习题 73 中那样选择。当长度 $\geqslant j$ 的一个键码头一次出现时，对于 h_j 生成随机系数（而且如果愿意，预先计算它的值的阵列）。由于 i 是无限的，因而矩阵 H 是无穷的。但是在程序的任何特定的运行中仅仅有限部分是有关的。

77.设 $p\leqslant 2^{-16}$ 是在 H 之下两个 32 个二进位键码有相同的映像的概率。当两个给定的键码在它们的八个 32 位二进位子键码的七个上一致时，则出现最坏情况；则冲突的概率是 $1-(1-p)^4<4p$。[参见 Wegman 和 Carter, *J. Comp. Syst. Sci.* **22** (1981), 265～279。]

6.5 节

1.提示中所描述的通路可以这样来加以转化，即把从 $(i-1,j)$ 运行到一个"新的最低记录"的值 $(i,j-1)$ 的每个往下的步骤改变成一个往上的步骤。如果做了 c 个这样的修改，则这条通路在 $(m,n-2t+2c)$ 处结束，其中 $c\geqslant 0$ 和 $c\geqslant 2t-n$；因此 $n-2t+2c\geqslant n-2k$。在与被修改的通路相对应的排列中，表列 B 的最小的 c 个元素对应于被修改的诸向下步骤，而表列 A 包含 $t-c$ 个对应于未修改的诸向下步骤。

当 $t=k$ 时，不难看出这个构造是可逆的；因此恰恰构造了 $\binom{n}{k}$ 个排列。附带地说，按照这个证明，表列 A 和 C 的内容可以以任意的次序出现。

注意：我们已经在习题 2.2.1-4 中以另一种方式计算了这些通路。当 $k=\lfloor n/2\rfloor$ 时这个构造证明了 Sperner 引理，它指出，不可能有 $\{1,2\cdots,n\}$ 的 $\binom{n}{\lfloor n/2\rfloor}$ 个以上的子集，使得这些子集中没有一个被包含于另一个当中。[Emanuel Sperner, *Math. Zeitschrift* **27** (1928), 544～548]。因为，如果我们有这样一个子集的集合，则 $\binom{n}{k}$ 个

排列中的每一个的初始位置上至多可以出现这些子集之一,但每个子集又应出现在某个排列中。这里所用的构造是一个更为一般的构造的改头换面的形式。N. G. de Bruijn C. van Ebbenhorst Tengbergen 和 D. Kruyswijk[*Nieuw Archief voor Wiskunde* (**2**)**23** (1951),191~193]通过这个构造,证明了 Sperner 的引理对多重集合的推广:"设 M 是包含有 n 个元素(计算了多重性的)一个多重集合。M 的所有$\lfloor n/2 \rfloor$个元素的子多重集合的全体,是使得没有任何子多重集合包含在另一个当中的最大可能的这种集合。"例如,当 $M=\{a,a,b,b,c,c\}$时,最大的这样的集合是由七个子多重集合$\{a,a,b\}$,$\{a,a,c\}$,$\{a,b,b\}$,$\{a,b,c\}$,$\{a,c,c\}$,$\{b,b,c\}$,$\{b,c,c\}$组成的。这将对应于六个属性 A_1,B_1,A_2,B_2,A_3,B_3 的七种排列,在这些属性中,所有涉及 A_i 的询问也涉及 B_i。进一步的评述出现在 C. Greene 和 D. J. Kleitman 的论文,*J. Combinatorial* **A20** (1976),80~88 中。

2. 令 a_{ijk} 为对一些记录的所有访问的表列,这些记录分别以(i,j,k)为三个属性的值,并假定 a_{011}是三个表列 a_{011},a_{101},a_{110}之最短者。则一个极小长度的表列是 $a_{001}a_{011}a_{111}a_{101}a_{100}a_{110}a_{111}a_{011}a_{010}$。然而,如果 a_{011}是空的,而且 a_{001},a_{010}或 a_{100}之一也是,则这个长度可以通过删去 a_{111}的两个出现之一来缩短[*CACM* **15** (1972),802~808]。

3. (a) 茴香籽和/或蜂蜜,可能同豆蔻和/或香精组合。(b) 无。

4. 设 p_t 是询问恰恰涉及 t 个二进位位置的概率,且 P_t 是在一个随机记录中给定的 t 个位置全都是 1 的概率。则答案是$\sum_t p_t P_t$ 减去一个特定的记录是一个"真情报"的概率,后者是$\binom{N-q}{r-q}\Big/\binom{N}{r}$,其中 $N=\binom{n}{k}$。由容斥原理

$$P_t = \sum_{j\geqslant 0}(-1)^j\binom{t}{j}f(n-j,k,r)/f(n,k,r)$$

其中 $f(n,k,r)$是在一个 n 位字段中,选择 r 个不同的 k 位属性码的可能的数目,即

$$f(n,k,r) = \binom{\binom{n}{k}}{r}$$

而且由习题 1.3.3-26,如果 $q=r$,

$$p_t = \sum_{l\geqslant 0}(-1)^l\binom{t+l}{t}\binom{n}{t+l}P_{t+l} =$$

$$\binom{n}{t}\sum_{j\geqslant 0}(-1)^j\binom{t}{j}f(t-j,k,q)/f(n,k,q)$$

注意:上述计算的更一般的形式,首先是由 G. Orosz 和 L. Takács 进行的,见 *J. of Documentation* **12** (1956),231~234。均值$\sum tp_t$ 容易证明是 $n(1-f(n-1,k,q)/f(n,k,q))$。另一个假定,即记录中和询问中的随机属性代码不必是不同的,如同在 Harrison 和 Bloom 的技术中那样,可以通过同样的方法来进行分析,并置 $f(n,k,$

$r)=\binom{n}{k}^r$。当参数在适当的范围中时，我们有 $P_t\approx(1-e^{-kr/n})^t$ 而且 $\sum p_tP_t\approx P_{n(1-\exp(-kq/n))}$。

6. $L(t)=\sum_j\binom{m_1}{j}\binom{m_2}{t-j}L_1(j)L_2(t-j)\Big/\binom{m_1+m_2}{t}$。[因此如果 $L_1(t)\approx N_1\alpha^t$ 而且 $L_2(t)\approx N_2\alpha^{-t}$，则 $L(t)\approx N_1N_2\alpha^{-t}$。]

7. (a) $L(1)=3,L(2)=1\frac{3}{4}$。(b) $L(1)=3\frac{3}{4},L(2)=2\frac{1}{3},L(3)=1\frac{9}{16}$。[注：诸如 00**→0，01**→1，10**→2，11**→3 这样一个平凡的投影映像有糟糕的最坏情况特性；但它有一个较好的平均情况，因为由习题得出 $L(1)=3,L(2)=2\frac{1}{6},L3=1\frac{1}{2}$。]

8. (a)当 $S=S_00\cup S_11$ 时，我们有 $f_t(S)=f_t(S_0\cup S_1)+f_{t-1}(S_0)+f_{t-1}(S_1)$。因此 $f_t(s,m)$是对于使得 $2^{m-1}\geqslant s_0\geqslant s_1\geqslant 0$ 和 $s_0+s_1=s$ 的所有 s_0 和 s_1，$f_t(s_0,m-1)+f_{t-1}(s_0,m-1)+f_{t-1}(s_1,m-1)$的极小值。为了证明对于 $s_0=\lceil s/2\rceil$ 和 $s_1=\lfloor s/2\rfloor$，出现极小值，我们可对 m 使用归纳法，对于 $m=1$ 结果是显然的：给定 $m\geqslant 2$，命 $g_t(s)=f_t(s,m-1)$和 $h_t(s)=f_t(s,m-2)$。于是，由归纳法，$g_t(s_0)+g_{t-1}(s_0)+g_{t-1}(s_1)=h_t(\lceil s_0/2\rceil)+h_{t-1}(\lceil s_0/2\rceil)+h_{t-1}(\lfloor s_0/2\rfloor)+h_{t-1}(\lceil s_0/2\rceil)+h_{t-2}(\lceil(s_0/2\rceil)+h_{t-2}(\lfloor s_0/2\rfloor)+h_{t-1}(\lceil s_1/2\rceil)+h_{t-2}(\lceil s_1/2\rceil)+h_{t-2}(\lfloor s_1/2\rfloor)$，它是$\geqslant g_t(\lceil s_0/2\rceil+\lceil s_1/2\rceil)+g_{t-1}(\lceil s_0/2\rceil+\lceil s_1/2\rceil)+g_{t-1}(\lfloor s_0/2\rfloor+\lfloor s_1/2\rfloor)$。而且如果 $s_0>s_1+1$，我们有$\lceil s_0/2\rceil+\lceil s_1/2\rceil<s_0$，除非在 $s_0=2k+1$ 和 $s_1=2k-1$ 的情况下。然而，在后一种情况下，$g_t(s_0)+g_{t-1}(s_0)+g_{t-1}(s_1)\geqslant h_t(2k+1)+2h_{t-1}(2k)\geqslant h_t(2k)+2h_{t-1}(2k)$。

(b)注意，包含在二进记法下的数 $0,1,\cdots,s-1$ 的集合 S 有如下性质：$S_0\cup S_1=S_0$ 而且 S_0 包含$\lceil s_0/2\rceil$个元素。顺便指出，由此得出，$f_t(2^{m-n},m)=[z^t](1+z)^n(1+2z)^{m-n}$。

10. (a) 必有$\frac{1}{6}v(v-1)$个三元组，而且 x_v 必然出现在它们的$\frac{1}{2}v$ 个当中。(b)因为 v 是奇数，对于每个 i 有惟一的三元组$\{x_i,y_j,z\}$，所以 S'容易证明是一个 Steiner 三元组系统。在 K'中不出现的对偶是$\{z,x_2\},\{x_2,y_2\},\{y_2,x_3\},\{x_3,y_3\},\cdots,\{x_{v-1},y_{v-1}\},\{y_{v-1},x_v\},\{x_v,z\}$。(d)由情况 $v=1$ 开始，应用操作 $v\to 2v-2$，$v\to 2v+1$ 即得不具有 $3k+2$ 形式的所有非负数，因为 $6k+(0,1,3,4)$诸情况分别地来自较小的情况 $3k+(1,0,1,3)$。

顺便指出，"Steiner 三元组系统"实际不应该取 Steiner 这个名称，尽管这一名称在文献中已打上深深的烙印。Steiner 的论著[*Crelle* **45** (1853)，181～182]出现在 Kirkman 的论著之后好多年，而且 Felix Klein 已经指出[*Vorlesungen über die En-*

twicklung der Math. im 19, Jahrhundert **1** (Springer, 1926), 128], Steiner 在他的晚年期间，引用了英国一些作者的话，而不提及他们。而且，这个概念已经出现在 J. Plücker 著名的两本书[*System der analytischen Geometrie* (1835), 283～284, *Theorie der algebraischen Curven* (1839), 245～247]中。

11. 取 $2v+1$ 个对象上的 Steiner 三元组系统。称诸对象之一为 z，而且以这样一个方式来命名其余对象，即包含 z 的三元组是 $\{z, x_i, \overline{x_i}\}$；删去这些三元组。

12. 对于 $0\leqslant k<14$，$\{k, (k+1) \bmod 14, (k+4) \bmod 14, (k+6) \bmod 14\}$，其中 $(k+7) \bmod 14$ 是 k 的补。[补系统是所谓可除群区组设计的特殊情况；参见 Bose, Shrikhande 和 Bhattacharya, *Ann. Math. Statistics* **24** (1953), 167～195。]

14. 在 $k-d$ 树中删除是最容易的(对根的替换可以在大约 $O(N^{1-1/k})$ 步之内求得)。在四叉树中，删除似乎要求重新构造以被删除的节点为根的整棵子树(但平均说来这个子树仅含大约 $\log N$ 个节点)在邮局树中，删除几乎毫无希望。

16. 设每一三元组对应于一个码字，其中每个码字恰有三个 1 的二进位，这些二进位标识对应的三元组的元素。如果 u, v, w 是不同的码字，v 与 w 的叠加至多与 u 有两个公共的二进位 1，因为它与单个的 v 或 w 至多有一位相同。[类似地，从阶为 v 的四元组系统我们可以构造 $v(v-1)/12$ 个码字，它们中没有一个被包含在任何其余三个的叠加当中，等等。]

17. (a)设对于 $1\leqslant k\leqslant n$，$c_0=b_0$，设 $c_k=$(如果 $b_{k-1}=0$ 则 $*$ 否则 b_k)，$c_{-k}=$(如果 $b_{k-1}=1$ 则 $*$ 否则 b_k)。则基本查询 $c_{-n}\cdots c_0\cdots c_n$ 描述筐 $b_0\cdots b_n$ 的内容。[因此这个方案是组合散列的一个特殊情况，而且它的平均查询时间匹配习题 8(b)中的下限。]

(b) 设对于 $-n\leqslant k\leqslant n$，$d_k=$[二进位 k 是确定的]。对于 $1\leqslant k\leqslant n$，我们可以假定 $d_{-k}\leqslant d_k$。则当确定的二进位全为 0 时，出现被考察的极大筐数，而且可以如下来计算它：置 $x\leftarrow y\leftarrow 1$。然后对于 $k=n, n-1, \cdots, 0$，置 $(x, y)\leftarrow(x, y)M_{d_{-k}+d_k}$，其中

$$M_0=\begin{pmatrix}1 & 1\\ 1 & 1\end{pmatrix},\ M_1=\begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix},\ M_2=\begin{pmatrix}1 & 1\\ 0 & 0\end{pmatrix}$$

最后输出 x(在 $k=0$ 之后，它也刚好等于 y)。

我们说，如果 $x\geqslant x'$ 和 $x+y\geqslant x'+y'$，则 $(x, y)\succeq(x', y')$。于是如果 $(x, y)\succeq(x', y')$，则我们有，对于 $d=0, 1, 2$，$(x, y)M_d\succeq(x', y')M_d$。现在

$$\begin{aligned}(x, y)M_2M_1^jM_0 &= (F_{j+3}x, F_{j+3}x)\\ (x, y)M_1M_1^jM_1 &= (F_{j+3}x+F_{j+2}y, F_{j+2}x+F_{j+1}y)\\ (x, y)M_0M_1^jM_2 &= (F_{j+2}x+F_{j+2}y, F_{j+2}x+F_{j+2}y)\end{aligned}$$

因此我们有 $(x, y)M_1M_1^jM_1\succeq(x, y)M_2M_1^jM_0$，因为 $2y\geqslant x$。而且类似地 $(x, y)M_1M_1^jM_1\succeq(x, y)M_0M_1^jM_2$，因为 $x\geqslant y$。由此得出，当对于 $1\leqslant k\leqslant n$，$d_{-k}+d_k\leqslant 1$ 或者当对于 $1\leqslant k\leqslant n$，$d_{-k}+d_k\geqslant 1$ 时，出现最坏情况。我们也有

$$(x,y)M_0M_1^j=(F_{j+2}x+F_{j+2}y,F_{j+1}x+F_{j+1}y)$$
$$(x,y)M_1^jM_0=(F_{j+2}x+F_{j+1}y,F_{j+2}x+F_{j+1}y)$$
$$(x,y)M_2M_1^j=(F_{j+2}x,F_{j+1}x)$$
$$(x,y)M_1^jM_2=(F_{j+1}x+F_jy,F_{j+1}x+F_jy)$$

因此,最坏情况要求下列的筐数:

$2^{n-t}F_{t+3}$	如果 $0\leqslant t\leqslant n$	[从 $M_1^tM_0^{n+1-t}$]
$2^{t-n}F_{3n-2t+3}$	如果 $n\leqslant t\leqslant\lceil 3n/2\rceil$	[从 $M_1^{3n-2t}(M_1M_2)^{t-n}M_0$]
2^{2n+1-t}	如果$\lceil 3n/2\rceil\leqslant t\leqslant 2n$	[从 $M_2^{2t-3n}(M_1M_2)^{2n-t}M_0$]

[这些结果实质上归功于 W. A. Burkhard, *BIT* **16** (1976), 13～31, 在 *J. Comp. Syst. Sci.* **15** (1977), 280～299 上又作了推广;但 Burkhard 从 $a_0\cdots a_{2n}$ 到 $b_0\cdots b_n$ 的更复杂的映射,如同由 P. Dubost 和 J. - M. Trousse Report STAN - CS - 75 - 511(斯坦福大学,1975)所建议的那样,这里已作了简化。]

18. (a) 在一起有 $2^n(m-n)$ 个 *,因此有 2^nn 个数字,且每列有 $2^nn/m$ 个数字。在每列中的半数的数字必然为 0。因此 $2^{n-1}n/m$ 是一个整数,而且每列含 $(2^{n-1}n/m)^2$ 个不匹配。由于每两行至少有一个不匹配,因此我们必定有 $2^n(2^n-1)/2\leqslant(2^{n-1}n/m)^2m$。

(b) 考虑在 $m-n$ 个特定的列为 0 的 2^n 个 m 个二进位的数。这些中的一半有奇校验。在任何未确定的列中有 * 的一行覆盖的奇数和偶数一样多。

(c) *000, *111, 0*10, 1*10, 00*1, 10*1, 010*, 110*。这一个不像(13)那样一致,因为像 *01* 这样的查询击中四个行而 *10* 仅仅击中两行。注意(13)有循环的对称性。

(d) 通过以 * * * * 来代替每个 *,以头四行的任何一行代替每个 0,以及以最后四行的任何一行来代替每个 1,从(13)的每行来建立 4^3 个行。(一个类似的构造从任何 ABD(m,n)和 ABD(m',n')构造一个 ABD(mm',nn')。

(e) 给定一个 ABD(16,9),我们可以以这样一种方式在每行中对一个 * 划圈,即使得每个列中有同样多的圆圈。然后我们可以把每行分成为两行,且以 0 和 1 来代替被划圈的元素。为了证明这样的划圈是可能的,注意每列的星号可以被任意地划分成 32 个组,每组含 7 个星号;于是 512 行的每个含 7 个不同组的星号,而且 $32\times16=512$个组,每组出现于 7 个不同的行中。定理 7.5.1E(“结婚定理”)现在保证一个完全匹配的存在性,且在每行和每组中恰好有一个被划圈的元素。

参考文献:R. L. Rivest, *SICOMP* **5** (1976), 19～50; A. E. Brouwer, *Combinatorics*, 由 Hajnal 和 Sós 主编, *Colloq Math. Soc. János Bolyai* **18** (1978), 173～184。对于所有 $n\geqslant32$, Brouwer 继续证明 ABD($2n,n$)存在。当把(13)同(15)组合在一起时,部分(d)的方法,也产生 ABD(32,15)。

19. 由习题 8,有 $8-k$ 个特定二进位的平均数是 $2^{k-3}f_{8-k}(8,8)/\binom{8}{k}$,对于 $8\geqslant$

$k \geqslant 0$,它有分别的值$(32,22,\frac{104}{7},\frac{69}{7},\frac{45}{7},\frac{33}{8},\frac{73}{28},\frac{13}{8},1) \approx (32,22,14.9,9.9,6.4,4.1,2.6,1.6,1)$。这些值仅仅稍微高于 $32^{k/8}$的值(32,20.7,13.5,8.7,5.7,3.7,2.4,1.5,1)。最坏情况的值是(32,22,18,15,11,8,4,2,1)。

20. J. A. La Poutre[*Disc. Math.* **58** (1986),205～208] 证明,当 $m > \binom{n}{2}$和 $n > 3$ 时 ABD(m,n)不可能存在;因此无 ABD(16,6)存在。La Poutre 和 van Lint[*Util. Math.* **31** (1987),219～225]证明,没有 ABD(10,5)。利用习题 18 的方法,从一个 ABD(8,5)或 ABD(4,3),我们得到一个 ABD(8,6);这产生出若干非自同构的解,而且附加的 ABD(8,6)的例子也存在。(除了平凡的 ABD(5,5)和 ABD(6,6)之外),剩下的仅有的可能性是,不同于(15)的 ABD(8,5),而且也许还有一个或更多的 ABD(12,6)。

All right—I'm glad we found it out detective fashion;
I wouldn't give shucks for any other way.
好吧—我很高兴我们以侦察的方式发现了它,
我对于任何其它的途径也将不会放弃。
——TOM SAWYER(1884)

附录 A　数值数量表

表 1　在标准子程序和在对计算机程序的分析中经常使用的数量(到小数点后 40 位)

$\sqrt{2}$	= 1.41421 35623 73095 04880 16887 24209 69807 85697−
$\sqrt{3}$	= 1.73205 08075 68877 29352 74463 41505 87236 69428+
$\sqrt{5}$	= 2.23606 79774 99789 69640 91736 68731 27623 54406+
$\sqrt{10}$	= 3.16227 76601 68379 33199 88935 44432 71853 37196−
$\sqrt[3]{2}$	= 1.25992 10498 94873 16476 72106 07278 22835 05703−
$\sqrt[3]{3}$	= 1.44224 95703 07408 38232 16383 10780 10958 83919−
$\sqrt[4]{2}$	= 1.18920 71150 02721 06671 74999 70560 47591 52930−
$\ln 2$	= 0.69314 71805 59945 30941 72321 21458 17656 80755+
$\ln 3$	= 1.09861 22886 68109 69139 52452 36922 52570 46475−
$\ln 10$	= 2.30258 50929 94045 68401 79914 54684 36420 76011+
$1/\ln 2$	= 1.44269 50408 88963 40735 99246 81001 89213 74266+
$1/\ln 10$	= 0.43429 44819 03251 82765 11289 18916 60508 22944−
π	= 3.14159 26535 89793 23846 26433 83279 50288 41972−
$1^\circ=\pi/180$	= 0.01745 32925 19943 29576 92369 07684 88612 71344+
$1/\pi$	= 0.31830 98861 83790 67153 77675 26745 02872 40689+
π^2	= 9.86960 44010 89358 61883 44909 99876 15113 53137−
$\sqrt{\pi}=\Gamma(1/2)$	= 1.77245 38509 05516 02729 81674 83341 14518 27975+
$\Gamma(1/3)$	= 2.67893 85347 07747 63365 56929 40974 67764 41287−
$\Gamma(2/3)$	= 1.35411 79394 26400 41694 52880 28154 51378 55193+
e	= 2.71828 18284 59045 23536 02874 71352 66249 77572+
$1/e$	= 0.36787 94411 71442 32159 55237 70161 46086 74458+
e^2	= 7.38905 60989 30650 22723 04274 60575 00781 31803+
γ	= 0.57721 56649 01532 86060 65120 90082 40243 10422−
$\ln \pi$	= 1.14472 98858 49400 17414 34273 51353 05871 16473−
ϕ	= 1.61803 39887 49894 84820 45868 34365 63811 77203+
e^γ	= 1.78107 24179 90197 98523 65041 03107 17954 91696+
$e^{\pi/4}$	= 2.19328 00507 38015 45655 97696 59278 73822 34616+
$\sin 1$	= 0.84147 09848 07896 50665 25023 21630 29899 96226−
$\cos 1$	= 0.54030 23058 68139 71740 09366 07442 97660 37323+
$-\zeta'(2)$	= 0.93754 82543 15843 75370 25740 94567 86497 78979−
$\zeta(3)$	= 1.20205 69031 59594 28539 97381 61511 44999 07650−
$\ln \phi$	= 0.48121 18250 59603 44749 77589 13424 36842 31352−
$1/\ln \phi$	= 2.07808 69212 35027 53760 13226 06117 79576 77422−
$-\ln \ln 2$	= 0.36651 29205 81664 32701 24391 58232 66946 94543−

表 2　在标准子程序和在对计算机程序的分析中经常使用的数量(到八进制 45 位)

0.1 =	0.06314 63146 31463 14631 46314 63146 31463 14631 46315 −
0.01 =	0.00507 53412 17270 24365 60507 53412 17270 24365 60510 −
0.001 =	0.00040 61115 64570 65176 76355 44264 16254 02030 44672 +
0.0001 =	0.00003 21556 13530 70414 54512 75170 33021 15002 35223 −
0.00001 =	0.00000 24761 32610 70664 36041 06077 17401 56063 34417 −
0.000001 =	0.00000 02061 57364 05536 66151 55323 07746 44470 26033 +
0.0000001 =	0.00000 00153 27745 15274 53644 12741 72312 20354 02151 +
0.00000001 =	0.00000 00012 57143 56106 04303 47374 77341 01512 63327 +
0.000000001 =	0.00000 00001 04560 27640 46655 12262 71426 40124 21742 +
0.0000000001 =	0.00000 00000 06676 33766 35367 55653 37265 34642 01627 −
$\sqrt{2}$ =	1.32404 74631 77167 46220 42627 66115 46725 12575 17435 +
$\sqrt{3}$ =	1.56663 65641 30231 25163 54453 50265 60361 34073 42223 −
$\sqrt{5}$ =	2.17067 36334 57722 47602 57471 63003 00563 55620 32021 −
$\sqrt{10}$ =	3.12305 40726 64555 22444 02242 57101 41466 33775 22532 +
$\sqrt[3]{2}$ =	1.20505 05746 15345 05342 10756 65334 25574 22415 03024 +
$\sqrt[3]{3}$ =	1.34233 50444 22175 73134 67363 76133 05334 31147 60121 −
$\sqrt[4]{2}$ =	1.14067 74050 61556 12455 72152 64430 60271 02755 73136 +
$\ln 2$ =	0.54271 02775 75071 73632 57117 07316 30007 71366 53640 +
$\ln 3$ =	1.06237 24752 55006 05227 32440 63065 25012 35574 55337 +
$\ln 10$ =	2.23273 06735 52524 25405 56512 66542 56026 46050 50705 +
$1/\ln 2$ =	1.34252 16624 53405 77027 35750 37766 40644 35175 04353 +
$1/\ln 10$ =	0.33626 75425 11562 41614 52325 33525 27655 14756 06220 −
π =	3.11037 55242 10264 30215 14230 63050 56006 70163 21122 +
$1° = \pi/180$ =	0.01073 72152 11224 72344 25603 54276 63351 22056 11544 +
$1/\pi$ =	0.24276 30155 62344 20251 23760 47257 50765 15156 70067 −
π^2 =	11.67517 14467 62135 71322 25561 15466 30021 40654 34103 −
$\sqrt{\pi} = \Gamma(1/2)$ =	1.61337 61106 64736 65247 47035 40510 15273 34470 17762 −
$\Gamma(1/3)$ =	2.53347 35234 51013 61316 73106 47644 54653 00106 66046 −
$\Gamma(2/3)$ =	1.26523 57112 14154 74312 54572 37655 60126 23231 02452 +
e =	2.55760 52130 50535 51246 52773 42542 00471 72363 61661 +
$1/e$ =	0.27426 53066 13167 46761 52726 75436 02440 52371 03355 +
e^2 =	7.30714 45615 23355 33460 63507 35040 32664 25356 50217 +
γ =	0.44742 14770 67666 06172 23215 74376 01002 51313 25521 −
$\ln \pi$ =	1.11206 40443 47503 36413 65374 52661 52410 37511 46057 +
ϕ =	1.47433 57156 27751 23701 27634 71401 40271 66710 15010 +
e^{γ} =	1.61772 13452 61152 65761 22477 36553 53327 17554 21260 +
$e^{\pi/4}$ =	2.14275 31512 16162 52370 35530 11342 53525 44307 02171 −
$\sin 1$ =	0.65665 24436 04414 73402 03067 23644 11612 07474 14505 −
$\cos 1$ =	0.42450 50037 32406 42711 07022 14666 27320 70675 12321 +
$-\zeta'(2)$ =	0.74001 45144 53253 42362 42107 23350 50074 46100 27706 +
$\zeta(3)$ =	1.14735 00023 60014 20470 15613 42561 31715 10177 06614 +
$\ln \phi$ =	0.36630 26256 61213 01145 13700 41004 52264 30700 40646 +
$1/\ln \phi$ =	2.04776 60111 17144 41512 11436 16575 00355 43630 40651 +
$-\ln \ln 2$ =	0.27351 71233 67265 63650 17401 56637 26334 31455 57005 −

注：= 号左边的名字是以十进制给出的

在同排序和查找算法的分析相关联中，出现有没有普通名字的若干有趣的常数。在等式5.2.3-(19)和6.5-(6)以及在习题5.2.3-27，5.2.4-13，5.2.4-23，6.2.2-49，6.2.3-7，6.2.3-8，6.3-26 以及 6.3-26 中，这些常数已被计算到小数点后面40位。

表3 对于小的 n 值的调和数，伯努利数以及斐波那契数的值

n	H_n	B_n	F_n	n
0	0	1	0	0
1	1	−1/2	1	1
2	3/2	1/6	1	2
3	11/6	0	2	3
4	25/12	−1/30	3	4
5	137/60	0	5	5
6	49/20	1/42	8	6
7	363/140	0	13	7
8	761/280	−1/30	21	8
9	7129/2520	0	34	9
10	7381/2520	5/66	55	10
11	83711/27720	0	89	11
12	86021/27720	−691/2730	144	12
13	1145993/360360	0	233	13
14	1171733/360360	7/6	377	14
15	1195757/360360	0	610	15
16	2436559/720720	−3617/510	987	16
17	42142223/12252240	0	1597	17
18	14274301/4084080	43867/798	2584	18
19	275295799/77597520	0	4181	19
20	55835135/15519504	−174611/330	6765	20
21	18858053/5173168	0	10946	21
22	19093197/5173168	854513/138	17711	22
23	444316699/118982864	0	28657	23
24	1347822955/356948592	−236364091/2730	46368	24
25	34052522467/8923714800	0	75025	25
26	34395742267/8923714800	8553103/6	121393	26
27	312536252003/80313433200	0	196418	27
28	315404588903/80313433200	−23749461029/870	317811	28
29	9227046511387/2329089562800	0	514229	29
30	9304682830147/2329089562800	8615841276005/14322	832040	30

对于任意的 x，设 $H_x = \sum_{n\geqslant 1}\left(\frac{1}{n}-\frac{1}{n+x}\right)$，于是

$$H_{1/2} = 2 - 2\ln 2,$$
$$H_{1/3} = 3 - \tfrac{1}{2}\pi/\sqrt{3} - \tfrac{3}{2}\ln 3,$$
$$H_{2/3} = \tfrac{3}{2} + \tfrac{1}{2}\pi/\sqrt{3} - \tfrac{3}{2}\ln 3,$$
$$H_{1/4} = 4 - \tfrac{1}{2}\pi - 3\ln 2,$$
$$H_{3/4} = \tfrac{4}{3} + \tfrac{1}{2}\pi - 3\ln 2,$$
$$H_{1/5} = 5 - \tfrac{1}{2}\pi\phi^{3/2}5^{-1/4} - \tfrac{5}{4}\ln 5 - \tfrac{1}{2}\sqrt{5}\ln\phi,$$
$$H_{2/5} = \tfrac{5}{2} - \tfrac{1}{2}\pi\phi^{-3/2}5^{-1/4} - \tfrac{5}{4}\ln 5 + \tfrac{1}{2}\sqrt{5}\ln\phi,$$
$$H_{3/5} = \tfrac{5}{3} + \tfrac{1}{2}\pi\phi^{-3/2}5^{-1/4} - \tfrac{5}{4}\ln 5 + \tfrac{1}{2}\sqrt{5}\ln\phi,$$
$$H_{4/5} = \tfrac{5}{4} + \tfrac{1}{2}\pi\phi^{3/2}5^{-1/4} - \tfrac{5}{4}\ln 5 - \tfrac{1}{2}\sqrt{5}\ln\phi,$$
$$H_{1/6} = 6 - \tfrac{1}{2}\pi\sqrt{3} - 2\ln 2 - \tfrac{3}{2}\ln 3,$$
$$H_{5/6} = \tfrac{6}{5} + \tfrac{1}{2}\pi\sqrt{3} - 2\ln 2 - \tfrac{3}{2}\ln 3,$$

而且，一般地，当 $0<p<q$ 时(参见习题 1.2.9-19)，

$$H_{p/q} = \frac{q}{p} - \frac{\pi}{2}\cot\frac{p}{q}\pi - \ln 2q + 2\sum_{1\leqslant n<q/2}\cos\frac{2pn}{q}\pi\cdot\ln\sin\frac{n}{q}\pi。$$

附录B 符号索引

在下列公式中,未被进一步定性的字母有如下的意义:

j, k	整数值算术表达式
m, n	非负整数值表达式
x, y	实数值算术表达式
z	复数值算术表达式
f	实数值或复数值函数
p	指针值表达式(或者是 Λ,或者是一个计算机地址)
S, T	集合或多重集合
α, β	符号串

形式符号	意　义	定义的位置
$V \leftarrow E$	把表达式 E 的值赋给变量 V	1.1
$U \leftrightarrow V$	交换变量 U 和 V 的值	1.1
A_n 或 $A[n]$	线性数组 A 的第 n 个元素	1.1
A_{mn} 或 $A[m, n]$	矩形数组 A 的 m 行 n 列的元素	1.1
NODE(P)	假定 P≠Λ,其地址为 P 的节点(由它们的字段名个别地区分的变量组)	2.1
F(P)	其字段名为 F 的 NODE(P)中的变量	2.1
CONTENTS(P)	其地址为 P 的计算机字的内容	2.1
LOC(V)	在一台计算机内变量 V 的地址	2.1
P⇐AVAIL	把指针变量 P 的值置成一个新节点的地址	2.2.3
AVAIL⇐P	把 NODE(P)恢复成自由存储;它的所有字段失去标识	2.2.3
top(S)	在一非空栈 S 顶部的节点	2.2.1
X⇐S	把 S 弹到 X:即置 $X \leftarrow$top(S);然后从非空栈 S 删去 top(S)	2.2.1
S⇐X	把 X 压入栈 S;把值 X 插入作为栈 S 顶部的新栈元	2.2.1
$(B \Rightarrow E; E')$	条件表达式:如果 B 为真表示 E,B 为假为 E'	
$[B]$	条件 B 的特征函数$(B \Rightarrow 1; 0)$	1.2.3
δ_{kj}	克洛涅克迭塔函数$[j = k]$	1.2.6
$[z^n]g(z)$	幂级数 $g(z)$中 z^n 的系数	1.2.9
$\sum_{R(k)} f(k)$	使得变量 k 是一个整数且关系 $R(k)$为真的所有 $f(k)$之和	1.2.3

（续）

形式符号	意　　义	定义的位置
$\prod_{R(k)} f(k)$	使得变量 k 是一个整数且关系 $R(k)$ 为真的所有 $f(k)$ 之积	1.2.3
$\min_{R(k)} f_k$	使得变量 k 是一个整数且关系 $R(k)$ 为真的所有 $f(k)$ 之极小值	1.2.3
$\max_{R(k)} f(k)$	使得变量 k 是一个整数且关系 $R(k)$ 为真的所有 $f(k)$ 之极大值	1.2.3
$j \backslash k$	j 整除 k；$k \bmod j = 0$ 且 $j > 0$	1.2.4
$S \setminus T$	集合的差：$\{a \mid a$ 在 S 中且 a 不在 T 中$\}$	
$\gcd(j,k)$	j 和 k 的最大公因子（$j = k = 0 \Rightarrow 0$；$\max_{d \backslash j, d \backslash k} d$）	1.1
$j \perp k$	j 与 k 互素：$\gcd(j,k) = 1$.	1.2.4
A^T	矩阵数组 A 的转置 $A^T[j,k] = A[k,j]$	1.2.3
α^R	α 的左右颠倒	
x^y	x 的 y 次方（当 x 为正时）	1.2.2
x^k	x 的 k 次方；（$k \geqslant 0 \Rightarrow \prod_{0 \leqslant j < k} x$；$1/x^{-k}$）	1.2.2
$x^{\overline{k}}$	x 的 k 升阶乘：$\Gamma(x+k)/\Gamma(x) = (k \geqslant 0 \Rightarrow \prod_{0 \leqslant j < k}(x+j)$；$1/(x+y)^{\overline{-k}}$）	1.2.5
$x^{\underline{k}}$	x 的 k 降阶乘：$x!\ (x-k)!\ = (k \geqslant 0 \Rightarrow \prod_{0 \leqslant j < k}(x-j)$；$1/(x-k)^{\underline{-k}}$	1.2.5
$n!$	n 的阶乘：$\Gamma(n+1) = n^{\underline{n}}$	1.2.5
$\binom{x}{k}$	二项式系数（$k < 0 \Rightarrow 0$；$x^{\underline{k}}/k!$）	1.2.6
$\binom{n}{n_1, n_2, \ldots, n_m}$	多项式系数（仅当 $n = n_1 + n_2 + \cdots + n_m$ 时才有定义）	1.2.6
$\left[{n \atop m}\right]$	第一类斯特林数：$\sum_{0<k_1<k_2<\cdots<k_{n-m}<n} k_1 k_2 \cdots k_{n-m}$	1.2.6
$\left\{{n \atop m}\right\}$	第二类斯特林数：$\sum_{1 \leqslant k_1 \leqslant k_2 \leqslant \cdots \leqslant k_{n-m} \leqslant m} k_1 k_2 \cdots k_{n-m}$	1.2.6
$\{a \mid R(a)\}$	使得关系 $R(a)$ 为真的所有 a 的集合	
$\{a_1, \ldots, a_n\}$	集合或多重集合 $\{a_k \mid 1 \leqslant k \leqslant n\}$	
$\{x\}$	分数部分（用于上下文中，指的是一个实数值而非集合）：$x - \lfloor x \rfloor$	1.2.11.2
$[a\,..\,b]$ 或 $[a,b]$	闭区间 $\{x \mid a \leqslant x \leqslant b\}$	1.2.2
$(a\,..\,b)$ 或 (a,b)	开区间 $\{x \mid a < x < b\}$	1.2.2
$[a\,..\,b)$ 或 $[a,b)$	半开区间：$\{x \mid a \leqslant x < b\}$	1.2.2
$(a\,..\,b]$ 或 $(a,b]$	半闭区间：$\{x \mid a < x \leqslant b\}$	1.2.2

（续）

形式符号	意　　义	定义的位置
$\|S\|$	基数：集合 S 中的元素个数	
$\|x\|$	x 的绝对值：$(x\geqslant0\Rightarrow x;-x)$	
$\|\alpha\|$	α 的长度	
$\lfloor x\rfloor$	x 的地板，即最大整数函数：$\max_{k\leqslant x}k$	1.2.4
$\lceil x\rceil$	x 的天花板，即最小整数函数：$\min_{k\geqslant x}k$	1.2.4
$x \bmod y$	mod 函数：$(y=0\Rightarrow x;x-y\lfloor x/y\rfloor)$	1.2.4
$x\equiv x'(\text{modulo } y)$	同余关系，$x \bmod y=x' \bmod y$	1.2.4
$O(f(n))$	当 $n\to\infty$ 时，$f(n)$的大 O	1.2.11.1
$O(f(z))$	当 $z\to0$ 时，$f(z)$的大 O	1.2.11.1
$\Omega(f(n))$	当 $n\to\infty$ 时，$f(n)$的大 Ω	1.2.11.1
$\Theta(f(n))$	当 $n\to\infty$ 时，$f(n)$的大 Θ	1.2.11.1
$\log_b x$	（当 $x>0,b>0$ 和 $b\neq1$ 时）x 的以 b 为底的对数，使得 $x=b^y$.	1.2.2
$\ln x$	自然对数：$\log_e x$	1.2.2
$\lg x$	二进制对数 $\log_2 x$	1.2.2
$\exp x$	x 的指数：e^x	1.2.2
$\langle X_n\rangle$	无穷序列 $X_0,X_1,X_2,\cdots$（这里字母 n 是符号的一部分）	1.2.9
$f'(x)$	f 在 x 处的导数	1.2.9
$f''(x)$	f 在 x 处的二阶导数	1.2.10
$f^{(n)}(x)$	n 阶导数：$(n=0\Rightarrow f(x);g'(x))$，其中 $g(x)=f^{(n-1)}(x)$	1.2.11.2
$H_n^{(x)}$	阶数为 x 的调和数 $\sum_{1\leqslant k\leqslant n}1/k^x$	1.2.7
H_n	调和数：$H_n^{(1)}$	1.2.7
F_n	斐波那契数：$(n\leqslant1\Rightarrow n;F_{n-1}+F_{n-2})$	1.2.8
B_n	贝努利数：$n!\,[z^n]z/(e^z-1)$	1.2.11.2
$\det(A)$	正方矩阵 A 的行列式	1.2.3
$\operatorname{sign}(x)$	x 的符号：$[x>0]-[x<0]$	
$\zeta(x)$	仄塔函数：$\lim_{n\to\infty}H_n^{(x)}$（当 $x>1$ 时）	1.2.7
$\Gamma(x)$	伽玛函数，$(x-1)!\;=\gamma(x,\infty)$	1.2.5
$\gamma(x,y)$	不完备的伽玛函数：$\int_0^y e^{-t}t^{x-1}dt$	1.2.11.3
γ	欧拉常数：$\lim_{n\to\infty}(H_n-\ln n)$	1.2.7
e	自然对数的底：$\sum_{n\geqslant0}1/n!$	1.2.2

（续）

形式符号	意　义	定义的位置
π	圆周率：$4\sum_{n\geq 0}(-1)^n/(2n+1)$	
∞	无穷大：比任何数都大	
Λ	空链（指向空地址的指针）	2.1
ϵ	空串（长度为0的串）	
$\emptyset$	空集（没有元素的集合）	
ϕ	黄金比：$\frac{1}{2}(1+\sqrt{5})$	1.2.8
$\varphi(n)$	欧拉的个数函数：$\sum_{0\leq k<n}[k\perp n]$	1.2.4
$x\approx y$	x 近似地等于 y	1.2.5
$\Pr(S(X))$	对于 X 的随机值，命题 $S(X)$ 为真的概率	1.2.10
$\mathrm{E}X$	X 的期望值：$\sum_x x\Pr(X=x)$	1.2.10
$\mathrm{mean}(g)$	由生成函数 g 表示的概率分布的平均值 $g^1(1)$	1.2.10
$\mathrm{var}(g)$	由生成函数 g 表示的概率分布的方差：$g''(1)+g'(1)-g'(1)^2$	1.2.10
$(\min x_1, \mathrm{ave}\ x_2, \max x_3, \mathrm{dev}\ x_4)$	有极小值 x_1，平均（期望值）x_2，极大值 x_3，标准差 x_4 的一个随机变量	1.2.10
$\Re z$	z 的实部	1.2.2
$\Im z$	z 的虚部	1.2.2
$\bar{z}$	复共轭 $\Re z - i\Im z$	1.2.2
$(\cdots a_1a_0.a_{-1}\cdots)_b$	b 进制位置记数符号：$\sum_k a_k b^k$	4.1
$/\!/x_1, x_2, \cdots, x_n/\!/$	连分数 $1/(x_1+1/(x_2+1/(\cdots+1/(x_n)\cdots)))$	4.5.3
$\alpha \top \beta$	相互穿插的积	5.1.2
$S\uplus T$	多重集合的和；例如 $\{a,b\}\uplus\{a,c\}=\{a,a,b,c\}$	4.6.3
$f(x)\|_a^b$	函数的增长：$f(b)-f(a)$	1.1
▌	算法，程序或证明的结束	
␣	一个空格	1.3.1
rA	MIX的寄存器A（累加器）	1.3.1
rX	MIX的寄存器X（扩充）	1.3.1
rI1，…，rI6	MIX的（变址）寄存器	1.3.1
rJ	MIX的（跳跃）寄存器	1.3.1
(L:R)	MIX的字的部分字段 $0\leq L\leq R\leq 5$	1.3.1

（续）

形式符号	意　　义	定义的位置
OP ADDRESS,I(F)	MIX 令的记号	1.3.1,1.3.2
u	MIX 中的时间单位	1.3.1
*	MIXAL 中的“自身”	1.3.2
0F,1F,2F,…,9F	MIXAL 中的“向前”局部符号	1.3.2
0B,1B,2B,…9B	MIXAL 中的“向后”局部符号	1.3.2
0H,1H,2H,…,9H	MIXAL 中的“这里”局部符号	1.3.2

人名和术语中英对照表

$-\infty$　$-\infty$负无穷大

1/3 − 2/3 conjecture　1/3 − 2/3 猜测

2-3 trees　2-3 叉树

(2,4) − trees　(2,4)树

2-d trees　2-d 树

2-descending sequence　2 递减序列

2-ordered permutations　2 有序排列

80-20 rule　80-20 定律

∞　∞无穷大

as sentinel　无穷大作为标记

π(circle ratio)　π(圆周率)

ϕ(golden ratio), xiv　ϕ(黄金分割比)

(a,b)-trees　(a,b)树

Abbreviated keys　缩写的键码

Abel, Niels Henrik, binomial formula　阿贝尔·尼尔斯·亨里克二项式公式

limit theorem　阿贝尔极限定理

Abraham, Chacko Thakadiparambil　阿布拉罕,查克戈·塔卡迪帕拉姆比尔

Absorption laws　吸收定律

Adaptive sorting　适配性排序

Addition of apples to oranges　苹果和桔子相加

Addition of polynomials　多项式加法

Addition to a list　加到一个表中,见 Insertion

Address calculation sorting　地址计算排序

Address table sorting　地址表排序

Adelson-Velsky, Georgii Maximovich　阿德尔森 − 维尔斯基,乔治·马克西莫维奇

Adjacent transpositions　相邻转置

Adversaries　对手

AF-heaps　AF − 堆

Agarwal, Ramesh Chandra　阿加尔瓦尔,拉米斯·钱德拉

Agenda,见 Priority queue　日程表

Aggarwal, Alok　阿加尔瓦尔,阿罗克

Aho, Alfred Vaino　阿霍,阿尔弗雷德·维诺

Aigner, Martin　埃泽纳尔,马丁

Ajtai, Miklós　阿伊泰,米克洛斯

al-Khwārizmī, Abū ʻAbd Alluāh Muhammad ibn Mūsā　阿尔-科瓦里兹末,阿布·阿布德·阿拉·穆哈默德·伊本·穆萨

Aldous, David John　——阿尔多斯,戴维·约翰

Alekseev, Vladimir Evgenievich　——阿历克谢耶夫,弗拉基米尔·叶南根尼维奇

Alexanderson, Gerald Lee　阿历山德森,杰拉德·李

AlGOL　——ALGOL(程序语言)

Algorithms, analysis of　算法分析,见 Analysis

comparison of　算法的比较,见 Comparison

proof of,算法的证明,见 Proof

Allen, Brian Richard　艾伦,布里安·理查德

Allen, Charles Grant Blairfindie　艾伦,查尔斯·格兰特·布赖尔芬迪

Alphabetic binary encoding　字母二进制编码

Alphabetic order　字母顺序

Altenkamp, Doris　阿尔登卡姆普,多利斯

Alternation runs　交错路段

Amble, Ole　安布,奥利

Amdahl, Gene Myron　阿姆达尔,洁恩·迈伦

American Library Association rules　美国图书馆学会规则

AMM: American Mathematical Monthly, published by the Mathematical Association of America since 1894.　自 1894 年以来由美国数学协会出版的美国数学月刊

Amortized cost　平摊的费用

Amphisbaenic sort　两端有头的排序

Anagrams,变异字,也见 Permutations of a multiset.

Analysis of algorithms　算法分析也见 Complexity analysis

Analytical Engine　分析机

AND (bitwise and)　与(按二进位进行的与运算)
André, Antoine Désiré　安德烈,安托万·德席里
Anti-stable sorting　反稳定排序
Antisymmetric function　反对称函数
Anuyogadvāra-sutra　阿鲁约加德瓦拉-苏特拉
Apollonius Sophista, son of Archibius　阿波洛尼厄斯·索菲斯塔,阿奇比尤斯的儿子
Appell, Paul Emile　阿普培尔,保罗·埃迈勒
Approximate equality　近似相等
Aragon, Cecilia Rodriguez　阿拉贡,塞西莉亚·罗德里格兹
Archimedes of Syracuse　赛拉古斯的阿基米德
　solids　阿基米德的立体
Arge, Lars Allan　阿尔基·拉斯·艾伦
Argument　自变量
Arisawa, Makoto　有泽诚
Arithmetic overflow　算术溢出
Arithmetic progressions　算术级数
Armstrong, Philip Nye　阿姆斯特朗·菲利普·奈
Arora, Sant Ram　阿罗拉·桑特·拉姆
Arpaci-Dusseau, Andrea Carol　阿尔帕西-德西奥,安德烈·卡洛尔
Arpaci-Dusseau, Remzi Hussein　阿尔帕西-德西奥,里米古·胡森
Ascents　递增
Ashenhurst, Robert Lovett　阿申赫斯特,罗伯特·洛维特
Askey, Richard Allen　阿斯基,理查德·阿伦
Associated Stirling numbers　结合的斯特林数
Assocative block designs　结合的区组设计
Associative law　结合律
Associative memories　联想存储
Asymptotic methods　渐近方法
　limits of applicability　渐近方法应用的极限
Attitude　态度
Attributes　属性
　binary　二进制属性
　compound　复合属性
auf der Heide, 奥弗·德·海德,见 Meyer auf der Heide

Automatic programming　自动程序设计
AVL trees, 459, AVL 树见 Balanced trees
Avni, Haim　艾夫尼·海伊姆

B-trees　B-树
B^+-trees　B^+树
B^*-trees　B^*树
Babbage, Charles　巴贝奇,查尔斯
Baber, Robert Laurence　巴伯,罗伯特·劳伦斯
Babylonian mathematics　巴比伦的数学
Bachrach, Ran　巴赫拉奇,兰
Backward reading　向后读,见 Read-backward
Baeza-Yates, Ricardo Alberto　巴伊扎-雅特斯,里加多·阿尔贝托
Bafna, Vineet　巴甫纳,维尼特
Bailey, Norman Thomas John　贝利,诺曼·托马斯·约翰
Balance factor　平衡因子
Balanced filing　平衡文件
Balanced incomplete block desigm　平衡的不完备区组设计
Balanced merging　平衡合并
with rewind overlap　通过重绕重迭的平衡合并
Balanced radix sorting　平衡的基数排序
Balaced trees　平衡树
　weight-balanced　权平衡的平衡树
Balancing a binary tree　平衡二叉树
Balancing a k-d tree　平衡 k-d 树
Balbine, Guy de　巴尔拜因,黄伊·笛
Ball, Walter William Rouse　鲍尔,沃尔特·威廉·劳斯
Ballot problem　投票问题
Barnett, John Keith Ross　巴尼特,约翰·基恩·罗斯
Barton, David Elliott　巴顿,戴维·埃里奥特
Barve, Rakesh Dilip　巴尔维,卡基恩·迪里普
Barycentric coordinates　重中心坐标
Basic query　基本查询
Batcher, Kenneth Edward　巴切尔,肯尼恩·爱德华
Batching　分批
Baudet, Gerard　鲍德特,杰勒德
Bayer, Paul Joseph　拜尔,保罗·约瑟夫

Bleier, Robert E.　布莱尔·罗伯特·E
Block designs　区组设计
Blocks of records　记录的块区
　on disk　盘上的块区
　on tape　带上的块区
Bloom, Burton H.　布鲁姆, 伯顿·H
Blum, Manuel　布鲁姆, 曼纽尔
Boas, Peter van Emde　波亚斯, 彼特·范·艾姆德
Boehme McGraw̄, Elaine M.　贝姆: 左格劳, 伊莱恩·M
Boerner, Hermann　博尔纳, 赫尔曼
Bollobás, Béla　波洛巴斯, 贝拉
Book of Creation　创世说
Boolean queries　布尔查询
Booth Andrew Donald　布恩, 安德鲁·唐纳德
Boothroyd, John　布恩罗伊德, 约翰
Borwein, Peter Benjamin　博尔威因, 彼得·本杰明
Bose, Raj Chandra　博斯, 拉伊·钱德拉
Bostic, Keith　博斯蒂克, 基恩
Bottenbruch, Hermann　伯登布鲁兹, 赫尔曼
Bouricius, Willard Gail　布里西尤斯, 威拉德·盖尔
Bourne, Charles Percy　查尔斯·珀西
Brandwood, Leonard　布兰德伍德, 里昂拉德
Brawn, Barbara Severa　布朗, 巴巴拉·西维拉
Breaux, Nancy Ann Eleser　布雷奥, 南希·安·埃里希
Brent, Richard Peirce　布伦特, 理查德·佩西
Briandais, René Edward de la　布廉戴斯, 雷内·爱德华·笛·拉
Brouwer, Andries Evert　布劳维尔, 安德里斯·埃弗特
Brown, John　布朗, 约翰
Brown, Mark Robbin　布朗, 马克·罗宾
Brown, Randy Lee　布朗, 兰迪·李
Brown, William Stanley　布朗, 威廉·斯坦利
Bruhat, Franççois, order　布鲁哈特, 弗朗索斯顺序
　weak　弱布鲁哈特顺序
Bruijn, Nicolaas Govert de　布鲁因, 尼古拉斯·哥威特·笛
Bryce, James Wares　布莱斯, 詹姆斯·沃里斯
Bubble sort　气泡排序
　multihead　多头气泡排序
Buchholz, Werner　布赫霍尔兹, 维纳
Buchsbaum, Adam Louis　布赫斯保姆, 阿当路易斯
Bucket sorting　桶排序
Buckets　桶
Buffering　缓冲
　size of buffers　缓冲区的大小
Bulk memory　海量存储, 见 Disk storage
Burge, William Herbert　伯奇, 威廉·赫伯特
Burkhard, Walter Austin　巴克哈德, 沃尔特·奥斯汀
Burton, Robert, v.　伯顿, 罗伯特
Butterfly network　蝴蝶网络

C　C程序语言
Cache memory　高速缓冲存储
CACM: Communications of the ACM, a publication of the Association for Computing Machinery since 1958　自1958年以来美国计算机器协会的一种出版物
Calendar queues　日历队
Cancellation laws　消去律
Canfield, Earl Rodney　坎菲尔德, 厄尔·洛德尼
Cards　卡片, 也见 Playing cards
　edge-notched　边带槽口的卡片
　feature　特征卡片
　machines for sorting　用于对卡片排序的机器
Carlitz, Leonard　卡里茨, 伦纳德
Caron, Jacques　卡伦, 雅克
Carries　进位
Carroll, Lewis (= Dodgson, Charles Lutwidge)　卡罗尔, 刘易斯 (= 道奇森, 查尔斯·勒特威奇)
Carter, John Lawrence　卡特, 约翰·劳伦斯
Carter, William Caswell　卡特, 威廉·卡斯威尔
Cartesian trees　笛卡儿树
Cascade merge　级联合并
　　read-backward　向后读级联合并
　　with rewind overlap　通过重统重迭的级联合并
Cascade numbers　级联树
Cascading pseudo-radix sort　级联虚拟基数排序
Catalan, Eugène Charles, numbers　卡塔兰, 尤金·查尔斯数

Fibonaccian search 斐波那契查找
Field,Daniel Eugene 菲尔德,丹尼尔·尤金
FIFO 先进先出,见 Queues
File:A sequence of records 文件:一个记录的系列
self-organizing 自组织的文件
Finding the maximum 寻找极大值
and minimum 寻找极大值和极小值
Fingers 手指
Finite fields 有限域
Finkel,Raphael Ari 芬克尔,拉菲尔·阿里
First-fit allocation 按"头一个适合"进行分配
First-in-first-out 先进先出,见 Queues
Fischer,Michael John 菲希尔,迈克尔·约翰
Fishburn,John Scot 菲什伯恩,约翰·斯科特
Fishspear 菲什皮尔
Fixed points of a permutation 一个排列的不动点
Flajoley,Philippe Patrick Michel 弗拉约勒特,菲利普·帕特里克·迈克尔
Flip operation 触发操作
Floating buffers 浮动缓冲区
Floating point accuracy 浮点精度
Flores,Ivan 弗洛里斯,伊凡
Floyd,Robert W 弗洛伊德,罗伯特,W
Foata,Dominique Cyprien 福塔,多米尼克·西普里安
FOCS:*Proceedings of the IEEE Symposia on Foundations of Computer Science* (1975 -), *formerly called the Symposia on Switching Circuit Theory and Logic Design* (1960 - 1965), *Symposia on Switching and Automata Theory* (1966-1974). 会议论文集
Folding a path 对折一条通路
Foldout illustration 折叠图解
Fomin,Sergey Vladimirovich 福明,塞奇·弗拉基米尔洛维奇
Ford,Donald Floyd 福特,唐纳德·弗洛伊德
Ford,Lester Randolph,Jr. 小福特,莱斯特·伦道夫
Forecasting 预报
Forest:Zero or more trees 森林:零个或多个树
FORTRAN FORTRAN 程序语言
Forward-testing-backward-insertion 向前测试向后插入
Foster,Caxton Croxford 福斯特,卡克斯顿·克劳福得
Fractal probability distribution 断片概率分布
Fractile insertion 碎片插入
Frame,James Sutherland 弗兰姆,詹姆斯·萨瑟兰
Françon,Jean 弗朗松,吉恩
Frank,Robert Morris 弗兰克,罗伯特·莫里斯
Franklin,Fabian 富兰克林,费比恩
Fraser,Christopher Warwick 弗雷泽,克里斯多弗·沃维克
Frazer,William Donald 弗雷泽,威廉·唐纳德
Fredkin,Edward 弗雷德金,爱德华
Fredman,Michael Lawrence 弗雷德曼,迈克尔·劳伦斯
Free groups 自由群
Free trees 自由树
Frequency of access 存取频率
Friedman,Haya 弗里德曼,哈雅
Friedman,Jerome Harold 弗里德曼.杰罗默·哈罗德
Friend,Edward Harry 弗兰德,爱德华·哈里
Frieze,Alan Michael 弗里泽,阿兰·迈克尔
Fringe analyses 边缘分析
Frobenius,Ferdinand Georg 弗罗比尼尤斯,弗迪南德·乔治
Front and rear compression 前后压缩
Fussenegger,Frank 弗森尼格,弗兰克

Gabow,Harold Neil 加保,哈罗德·尼尔
Gaines,Helen Fouché 盖恩斯,黑林·福彻
Gale,David 盖尔,戴维
Galen,Claudius 盖伦,克劳迪亚斯
Galil,Zvi 加里尔,泽维
Gamma function $\Gamma(z)$ 伽玛函数 $\Gamma(z)$
Gandz,Solomon 甘兹,所罗门
Gardner,Erle Stanley 加德纳,厄尔·斯坦利
Gardner,Martin 加德纳,马丁
Gardy,Danièle 加尔蒂,丹尼尔
Garsia,Adriano Mario 加西亚,阿德里亚诺·马里奥
Garsia-Wachs algorithm 加西亚-沃彻算法

Gasarch, William Ian 加沙尔兹，威廉·伊恩
Gassner, Betty Jane 加斯纳，贝蒂·简
Gaudette, Charles H. 高德特，查尔斯·H
Gauβ (= Gauss), Johann Friedrich Carl (= Carl Friedrich) 高斯，约翰·弗雷德里泽·卡尔
Gaussian integers 高斯整数
gcd: Greatest common divisor 最大公因子
Generable integer 可生成整数
Generating functions, techniques for using 使用生成函数的技术
Genes 杰尼斯
Genetic algorithms 遗传函数
Genoa, Giovanni di 吉诺亚，吉乔瓦尼·迪
Geometric data 几何数据
George, John Alan 乔治，约翰·阿伦
Gessel, Ira Martin 杰塞尔，伊拉·马丁
Ghosh, Sakti Pada 戈斯，萨克蒂·帕达
Gibson, Kim Dean 吉布森，金·迪安
Gilbert, Edgar Nelson 吉尔伯特，埃德加·纳尔逊
Gilbreath, Norman Laurence 吉尔布里思，诺尔曼·劳伦斯
principle 吉尔布里思原理
Gillis, Joseph 吉尔里斯，约瑟夫
Gilstad, Russell Leif 吉尔斯塔德，拉塞尔，莱夫
Gini, Gorrado 吉尼，科拉多
Gleason, Andrew Mattei 格里森，安德鲁·马太
Goetz, Martin Alvin 戈茨，马丁·阿尔文
Goldberg, Andrew Vladislav 戈德堡，安德鲁·弗拉基斯拉夫
Golden ratio, xiv 黄金分割比
Goldenberg, Daniel 戈登堡，丹尼尔
Goldstein, Larry Joel 戈尔斯坦，拉里·乔尔
Golin, Mordecai J. 戈林，莫尔德塞·J(高可龄)
Gonnet Haas, Gaston Henry 康内特，哈斯·加斯顿·亨利
Good, Irving John 古德，欧文·约翰
Goodman, Jacob Eli 古德曼，雅可布·埃里
Goodwin, David Thomas 古德温，戴维·托马斯
Gore, John K. 戈尔，约翰·K
Gotlieb, Calvin Carl 戈特利布，卡尔文·卡尔
Goto, Eiichi 后藤英一
Gourdon, Xavier Richard 戈尔顿，扎维尔·理查德
GPX system GPX 系统(UNIVAX 计算机中的一个系统)
Grabner, Peter Johannes 格拉布纳，彼得·约翰
Graham, Ronald Lewis 格雷厄姆，罗纳德·刘易斯(葛立恒)
Grasselli, Antonio 格拉塞利，安东尼奥
Grassl, Richard Michael 格拉塞尔，理查德·迈克尔
Gray, Harry Joshua, Jr. 格雷，哈里·小乔舒亚
Gray, James Nicholas 格雷，詹姆斯·尼古拉斯
Greatest common divisor 最大公因子
Green, Milton Webster 格林，米尔顿·韦伯斯特
Greene, Curtis 格林，柯蒂斯
Greene, Daniel Hill 格林，丹尼尔·希尔
Greniewski, Marek 格里纽斯基，马雷克
Grid files 栅格文件
Gries, David Joseph 格里斯，戴维·约瑟
Grinberg, Victor Simkhovich 格林贝格，维克多·西姆克霍维奇
Griswold, William Gale 格里斯沃德，威廉·加里
Gross, Oliver Alfred 格罗斯，奥里弗·艾尔弗雷德
Grossi, Roberto 格罗西，罗伯托
Group, free 自由群
Group divisible block designs 群可分区设计
Grove, Edward Franklin 格罗弗，爱德华·富兰克林
Growth ratio 生长率
Guibas, Leonidas John 吉巴斯，利奥尼达斯·约翰
Guilbaud, Georges Théodule 吉保德，乔治斯·西奥度尔
Gunji, Takao 郡司隆男
Gustafson, Richard Alexander 吉斯塔夫森，理查德·亚历山大
Gustavson, Frances Goertzel 吉斯塔夫森，弗朗西斯·戈策尔
Gwehenberger, Gernot 格温贝格，杰尔诺特
Gyrating sort 回旋排序

h-ordered sequence h-次序序列
Hadian, Abdollah 哈迪安，阿布多拉

Hajela, Dhananjay 哈杰拉,丹南贾伊
Hajnal, András 哈伊纳尔,安德拉斯
Half-balanced trees 半平衡树
Hall, Marshall, Jr 小霍尔,马歇尔
Halperin, John Harris 霍尔珀林,约翰·哈里斯
Halpern, Mark Irwin 霍尔珀林,马克·欧文
Hamilton, Douglas Alan 汉密尔顿,道格拉斯
Han, Guo-Niu 韩国牛
Hanan, Maurice 哈南,莫里斯
Hannenhalli, Sridhar Subrahmanyam 汉能哈里,斯里德哈·舒伯拉马尼亚姆
Haralambous, Yannis 哈拉兰保斯,雅尼斯
Hardy, Godfrey Harold 哈迪,戈弗雷·哈罗德
Hardy, Norman 哈迪,诺曼
Harmonic numbers 调和数
  generalized 广义调和数
Harper, Lawrence Hueston 哈珀,劳伦斯·休斯顿
Harrison, Malcolm Charles 哈里森,马尔科姆·查尔斯
Hash functions 散列函数
  combinatorial 组合散列函数
Hash sequences 散列序列
Hashing 散列
Havas, George 哈瓦斯,乔治
Hayward, Ryan Bruce 海沃德,里安·布鲁斯
Heap: A heap-ordered array 堆:一个堆编序的数组
Heap order 堆序
Heaps, Harold Stanley 哈罗德·斯坦利堆
Heapsort 堆排序
  with equal keys 有相等键码的堆排序
Heide 海德,见 Meyer auf der Heide
Height-balanced trees 高度平衡树
Height of extended binary tree 扩充的二叉树的高度
  of random binary search tree 随机二叉查找树的高度
  of random digital search tree 随机数字查找树的高度
  of random $(M+1)$-ary search tree 随机 $M+1$ 进制查找树的高度
  of random Patricia tree 随机帕特里西亚树的高度
  of random trie 随机检索结构的高度
Heilbronn, Hans Arnold 海尔布伦,汉斯·阿尔诺德
Heising, William Paul 海辛,威廉·保罗
Heller, Robert Andrew 赫勒,罗伯特·安德鲁
Hellerman, Herbert 赫勒曼,赫伯特
Hellerstein, Joseph Meir 赫勒斯坦,约瑟夫·梅尔
Hellman, Martin Edward 赫尔曼,马丁·爱德华
Hendricks, Walter James 亨德里克斯,沃尔特·詹姆斯
Hennequin, Pascal Daniel Michel Henri 亨尼奎因,巴斯卡·丹尼尔·迈克尔·亨利
Hennie, Frederick Clair 亨尼,弗里德里克·克莱尔
Hermite, Charles, polynomial 赫密特,查尔斯多项式
Herrick, Robert 赫里克,罗伯特
Hibbard, Thomas Nathaniel 希巴德,托马斯·纳撒尼尔
Hilbert, David 希巴德,戴维
Hildebrandt, Paul 希尔德布兰特,保罗
Hillman, Abraham P 希尔曼,阿布拉罕,P
Hindenburg, Karl Friedrich 欣登伯格,卡尔·弗里德里克
Hinrichs, Klaus Helmer 欣里彻斯,克劳斯·赫尔默
Hinterberger, Hans 欣登贝格,汉斯
Hinton, Charles Howard 欣顿,查尔斯·霍华德
Hoare, Charles Antony Richard 霍尔,查尔斯·安托尼·理查德
Hobby, John Douglas 霍比,约翰·道格拉斯
Hoey, Daniel J. 雷伊,丹尼尔·J
Holberton, Frances Elizabeth Snyder, 霍尔伯顿,弗朗西斯,伊丽莎白·斯奈德
Hollerith, Herman 霍勒里特,赫尔曼
Holt Hopfenberg, Anatol Wolf 霍尔特·霍普芬伯格,阿纳托尔·沃尔夫
Homer 霍默
Homogeneous polynomial 齐次多项式
Hooker, William Weston 胡克,威廉·韦斯顿
Hooking-up of queues 队的钩接
Hooks 钩
  generalized 扩充 61 钩
Hopcroft, John Edward 霍普克罗夫特,约翰·爱德华

Internal searching 内部查找
summary 内部查找小结
Internal sorting 内部排序
summary 内部排序小结
Internet 互联网
Interpolation search 内插查找
Interval-exchange sort 区间交换排序
Interval heap 区间堆
Intervals 区间
Inverse in a group 一个群中的逆
Inverse modulo m 模 m 之下的逆
Inverse of a permutation 一个排列的反序
for multisets 作为多重集合的排列的反序
Inversion tables of a permutation 一个排列的反序表
Inversions of a permutation 带有相等的一个排列的反序
with equality 一个排列的反序
Inversions of tree labelings 树标号的反序
Inverted files 反序文件
Involution coding 卷积编码
Involutions 卷积
Isaac, Earl J 伊萨克,厄尔·J
Isaiah son of Amoz 伊赛雅,阿莫兹之子
Isbitz, Harold 伊斯比兹,哈罗德
Isidorus of Seville, Saint(San Isidoro de Sevilla) 塞维利亚的圣·伊西多拉斯(圣·伊西多罗·德·塞维拉)
Ismail, Mourad El Houssieny 伊斯梅尔,毛拉德·厄尔·豪斯雪尼
Isomorphic invariants 同构不变量
Isomorphism testing 同构测试
Itai, Alon 伊泰·艾隆
Iverson, Kenneth Eugene 艾弗森,肯尼特·尤金

JACM: *Journal of the ACM*, a publication of the Association for Computing Machinery since 1954 1954 年以来计算机器协会的一个刊物
Jacobi, Carl Gustav Jacob, 雅各比,卡尔·吉斯塔夫·雅各布
Jacquet, Philippe Pierre 雅奎特,菲利普·皮埃尔
JAE(Jump if rA even) JAE(如果 rA 为偶数则跳转)
Jainism (印度的) 那数
Janson, Carl Svante 简森,卡尔·斯万特
JAD(Jump if rA odd) JAO(如果 rA 为奇数则跳转)
Jensen, Johan Ludvig William Valdemar 詹森,约翰,鲁德维格·威廉·瓦尔德马尔
John, John Welliaveetil 约梅,约翰·威里亚维蒂尔
Johnsen, Robert Lawrence 约翰逊,罗伯特·劳伦斯
Johnsen, Thorstein Lunde 约翰逊,索尔斯坦·伦德
Johnson, Lyle Robert 约翰逊,莱尔·罗伯特
Johnson, Selmer Martin 约翰逊,塞尔默·马丁
Johnson, Stephen Curtis 约翰逊,斯蒂芬·柯蒂斯
Johnson, Theodore Joseph 约翰逊,西奥多·约瑟夫
Joke 玩笑
Jonassen, Arne Tormod 乔纳森,阿恩·托莫德
Josephus, Flavius, son of Matthias 约瑟夫斯,弗雷维厄斯,马特蒂亚斯的儿子
problem 约瑟夫斯问题
Juillé, Hugues René 朱尔利,休古斯·里尼
Jump operators of MIX MIX 的跳转操作符
k-d trees k-d 树
k-d tries k-d 检索结构
Kaas, Robert 卡斯,罗伯特
Kabbala 卡布巴拉
Kaehler, Edwin Bruno 凯赫勒,埃德温·布鲁诺
Kalai 凯莱·吉尔
Kaman, Charles Henry 卡曼,查尔斯·亨利
Kant, Immanuel 坎特,伊姆曼纽尔
Kaplan, Aryeh 卡普兰,阿里耶
Kaplan, Haim 卡普兰,海姆
Karlin, Anna Rochelle 卡林,安纳·罗谢尔
Karp, Richard Manning 卡普,理查德·曼宁
Katajainen, Jyrki Juhani 卡塔贾依能,伊尔基·朱哈尼
Kaufman, Marc Thomas 考夫曼,马克·托马斯
Kautz, William Hall 考茨,威廉·霍尔
Kececioglu, John Dmitri 基西希奥格鲁,约翰·德密特里
Kelly, Wayne Anthony 凯利,韦恩·安托尼
Kemp, Rainer 肯珀,莱因纳
Kempner, Aubrey John 肯珀纳,奥布里·约翰

Kerov, Sergei Vasilievich　克洛夫，基尔盖·瓦西里耶维奇
Keys　键码
Keysorting　键码排序
Khizder, Leonid Abramovich　凯泽德尔，利奥尼德·阿布拉莫维奇
Kingston, Jeffrey H.　金斯顿，杰弗里·H
Kipling, Joseph Rudyard　基普林，约瑟夫·拉迪亚德
Kircher, Athanasius　柯切尔，阿塔纳修斯
Kirchhoff, Gustav Robert, first law,　柯希霍夫，古斯塔夫·罗伯特第一定律
Kirkman, Thomas Penyngton　柯克曼，托马斯·彭宁顿
　triple systems　柯克曼三元组系统
Kirkpatrick, David Galer　柯克帕特里克，戴维·盖勒
Kirschenhofer, Peter　克斯成霍弗，彼得
Kislitsyn, Sergei Sergeevich　基斯里特辛，塞盖·塞盖耶维奇
Klarner, David Anthony　克拉尔纳．戴维·安托尼
Klein, Christian Felix　克莱因，克里斯蒂安·菲利克斯
Kiein, Rolf　克莱因，罗尔夫
Kleitman, Daniel J (Isaiah Solomon)　克莱特曼，丹尼尔·J(伊塞雅·所罗门)
Klerer, Melvin　克莱尔，梅尔文
Knockout tournament　淘汰塞
Knott, Gary Don　克诺特，加里·唐
Knuth, Donald Ervin　克努特，唐纳德·欧文(高德纳)
Koch, Gary Grove　科克，加里·格罗夫
Koester, Charles Edward　凯斯特勒，查尔斯·爱德华
Köhler, Peter　科勒，彼得
Kollár, Lubor　科尔拉，卢博尔
Komlós, János　科姆罗斯，詹诺斯
Konheim, Alan Gustave　康海姆，艾伦·古斯塔夫
Koornwinder, Tom Hendrik　科恩温德，托姆·亨德里克
Korn, Granino Arthur　科恩，格拉尼诺·阿瑟
Körner, János　科恩纳，詹诺斯
Kreweras, Germain　克雷威拉斯，日尔曼因
Kronecker, Leopold　克罗尼克尔，利奥博德
Kronmal, Richard Aaron　克朗马尔，理查德·艾伦
Kronrod, Mikhail Aleksandrovich　克朗罗德，米克海尔·阿列克山德罗维奇
Krutar, Rudolph Allen　克鲁塔，鲁道夫，艾伦
Kruyswijk, Dirk　克鲁伊斯维克，迪克
Kummer, Ernst Eduard　库姆默，恩斯特·爱度华
Kwan, Lun　伦关
KWIC index　上下文中的键码索引

La Poutré, Johannes Antonius (= Han),　拉·保德利，约翰尼斯·安托尼尤斯(=汉)
Labelle, Gilbert　拉贝勒，吉尔伯特
Ladner, Richard Emil　拉德纳，理查德·埃密尔
Laforest, Louise　拉弗勒斯特，路易丝
Lagrange (= de la Grange), Joseph Louis, Comte, inversion formula　拉格朗日(=德·拉·格朗日)，约瑟夫·路易斯科姆特，反演公式
Laguerre, Edmond Nicolas, polynomials　拉奎尔，爱德蒙德·尼古拉斯多项式
LaMarca, Anthony George　拉马尔加，安托尼·乔治
Lambert, Johann Heinrich　拉姆伯特，约翰·亨利兹
　series　拉姆伯特序列
Lampson, Butler Wright　兰普森，巴特勒·赖特
Landauer, Walter Isfried　兰德，利昂·约瑟夫
Lander, Leon Joseph　兰多尔，沃尔特·艾斯弗里德
Landis, Evgenii Mikhailovich　兰迪斯，尤金尼·米克海伊罗维奇
Langston, Michael Allen　朗斯顿，迈克尔·艾伦
Lapko, Olga Georgievna　拉普戈，奥尔加·乔吉耶夫纳
Laplace (= de la Place), Pierre Simon, Marquis de　拉普拉斯(=德·拉·普拉斯)，皮埃尔·西蒙马尔奎伊斯·德
LARC Scientific Compiler　LARC 科学编译程序
Large deviations　大离差
Largest-in-first-out，最大者先出，见 Priority queues
Larson, Per-Ake　拉尔森，珀-阿基
Lascoux, Alain　拉斯科克斯·阿赖恩
Last-come-first-served　后来先服务
Last-in-first-out　后进先出
Latency time　等待时间

Latin language 拉丁语
Lattice,of bit vectors 二进位向量的格
  of permutations 排列的格
  of trees 树的格
lattice paths 格盘通路
Lawler,Eugene Leighton 劳勒,尤金·莱顿
Lazarus,Roger Ben 拉扎勒斯,罗杰·本
Least-recently-used page replacement 最近最少使用的页替换
Least-significant-digit-first radix sort 最低有效位数字优先基数排序
Leaves 叶
Lee,Der-Tsai 李德财
Lee,Tsai-hwa 李再华
Leeuwen,Jan van 刘文,简·万
Leeuwen,Marcus Aurelius Augustinus van 刘文,马库斯·奥里刘斯·奥古斯蒂奴斯
Lefkowitz,David 列莫科维兹,戴维
Left-to-right(or right-to-left)maxima or minima 自左向右(或自右向左)极大值或极小值
Leftist trees 左倾树
  deletion from 从左倾树删去
  insertion into 插入到左倾树
  merging 合并左倾树
Lehmer,Derrick Henry 莱默,德里克·亨利
Leibholz,Stephen Wolfgang 莱博霍尔兹,斯蒂芬·沃夫冈
Leiserson,Charles Eric 莱塞森,查尔斯·埃里克
Levcopoulos,Christos 利夫科保罗斯·克里斯托斯
Level of a tree node:The distance to the root 一个树节点的级:到根的距离
Levenshtein,Vladimir Iosifovich 利温斯坦,弗拉吉米尔·约西夫维奇
Leveque,William Judson 勒维克,威廉·朱丹
Levitt,Karl Norman 利维特,卡尔·诺尔曼
Levy,Silvio Vieira Forreira 利维,西尔维奥·弗雷拉
Lexicographic order 词典编辑顺序
lg:Binary logarithm lg:二进制对数
Li Shan-Lan 李善兰
Liang,Franklin Mark 梁,富兰克林·马克
Library card sorting 图书馆卡片排序
Liddy,George Gordon 利迪,乔治·戈顿
LIFO 后进先出,见 Stacks
Lin,Andrew Damon 林,安德鲁·达蒙
Lin,Shen 林甡
Lineal chart 直系图
Linear algorithm for median 求中值的线性算法
Linear algorithms for morting 排序的线性算法
Linear arrangements,optimum 最优线性安排
Linear congruential sequence 线性同余序列
Linear hashing 线性散列
Linear lists 线性表,也见 List sorting
  representation of 线性表的表示
Linear order 线性顺序
Linear probing 线性探查
  optimum 最优线性探查
Ling,Huei 林辉
Linial,Nathen 利尼尔,纳森
Linked allocation 链接分配
Linn,John Charles 林,约翰·查尔斯
Lint,Jacobus Hendricu van 林特,雅各布斯·亨德里库斯·万
Lipski,Witold,Jr. 小利普斯基,威托德
Lissajous,Jules Antoine 利沙佐乌斯,朱里斯·安托伊尼
List head 表头
List insertion sort 表插入排序
List merge sort 表合并排序
List sorting 表排序
Littlewood,Dudley Ernent, 利特尔伍德,达德利·欧内斯特
Littlewood,John Edenson 利特尔伍德,约翰·伊登索
Litwin,Samuel 利特文,萨缪尔
Livius,Titus 利维亚斯,泰特斯
Lloyd,Stuart Phinney 劳埃德,斯图尔特·芬尼
Load factor 负载因子
Load point 装入点
Logan,Benjamin Franklin(= Tex),Jr. 洛根,本杰明·富兰克林
Logarithmic search 对数查找

for rearranging　重新安排的极小空间算法
for selection　选择的极小空间算法
for sorting　排序的极小空间算法
for stable sorting　稳定排序的极小空间算法
Minimum-stage tape sorting　极小阶段磁带排序
Minimum-time algorithms　极小时间算法
for merging　合并的极小时间算法
for sorting　排序的极小时间算法
Minker, Jack　明克尔，杰克
MinuteSort　一分钟排序算法
Mises, Richard, Edler von　米塞斯，理查德·埃德勒
Misspelled names　拼借的名字
Mitchell, Oscar Howard　米特彻尔，奥斯卡·霍华德
MIX computer: A hypothetical machine defined in Section 1.3　MIX 计算机
MIXAL: The MIX assembly language　MIXAL: MIX 的汇编语言
MIXT tape units　MIXT 带设备
MIXTEC disks and drums　MIXTEC 磁盘和磁鼓
Miyakawa, Masahiro　宫川正弘
Möbius, August Ferdinand,　莫比乌斯，奥古斯特·费迪南德
function $\mu(n)$　莫比乌斯函数
Modified external path length　修改的外部通路长度
Moffat, Alistair　莫法特，阿里斯太尔
Molodowitch, Mariko　莫洛多维奇，马里戈
Monotonic subsequences　单调子序列
Monotonicity property　单调性的性质
Mönting　芒廷，见 Schulte Mönting
Moore, Edward Forrest　穆尔，爱德华·福雷斯特
Moore School of Electrical Engineering　穆尔电器工程学校
Morgenthaler, John David　莫根撒勒，约翰·戴维
Morris, Robert　莫里斯，罗伯特
Morrison, Donald Ross　莫里森，唐纳德·罗斯
Morse, Samuel Finley Breese, code　莫尔斯，基缪尔·芬利·布鲁斯代码
Mortenson, John Albert　莫顿森，约翰·阿尔伯特
Moser, Leo　莫泽，利奥
Most-significant-digit-first radix sort　最高有效位数字优先基数排序
Motzkin, Theodor Samuel　莫茨金，西奥多·基缪尔
Move-to-front heuristic　移动到前边带启发式的
MSD: Most significant digit　最高有效位数字
Muir, Thomas　米尔，托马斯
Mullin, James Kevin　马林，詹姆斯，克文
Multi-attribute retrieval　多属性检索，见 Secondary key retrieval
Multidimensional binary search trees　多维二叉查找树，见 k-d trees
Multidimensional tries　多维检索结构，见 k-d tries
Multihead bubble sort　多头气泡排序
Multikey quicksort　多键码快速排序
Multilist system　多表系统
Multinomial coefficients　多项式系数
Multiple list insertion sort　多表插入排序
Multiple-precision constants　多精度常数
Multiples of an irrational number mod 1　一个无理数在 1 之下模的多重性
Multiprecision comparison　多精度比较
Multiprocessing　多道处理
Multireel files　多卷筒文件
Multiset　多重集合：类似于一个集合，但元素可出现一次以上
ordering　对多重集合排顺序
permutations　多重集合的排列
sum and union　多重集合的和与并
Multivalued logic　多值逻辑
Multiway trees　多路树，也见 Tries
Multiword keys　多字键码
Munro, James Ian　芒罗，詹姆斯·伊恩
Muntz, Richard　穆恩茨，理查德
Muroga, Saburo　室贺三郎
Music　音乐
Musser, David Rea　墨基尔，戴维·里亚
Myers, Eugene Wimberly, Jr　小迈耶斯，尤金·温贝利

Nagler, Harry　纳格勒，哈里
Nakayama, Tadasi　中山正
Naor, Simeon(=Moni)　纳俄，西米昂(=莫尼)

Narasimhan, Balasubramanian 纳拉西姆罕·巴拉舒布拉马尼安
Natural correspondence between forests and binary trees 树林与二叉树之间的自然对应
Natural merge sort 自然合并排序
Natural selection 自然选择
Nearest neighbors 最接近的邻居
Needle 针
Negative links 负的链接
Neighbors of a point 一个点的邻居
Neimat, Marie-Anne Kamal 内马特，马里-安尼·卡马尔
Nelson, Raymond John 纳尔逊，雷蒙德·约翰
Netto, Otto Erwin Johannes Eugen 内特托，奥托·欧文·约翰尼斯·尤金
Networks of comparators, 比较器的网络
for merging 用于合并的比较器网络
for permutations 用于排列的比较器网络
for selection 用于选择的比较器网络
for sorting 用于排序的比较器网络
primitive 原始的比较器网络
standard 标准比较器网络
with minimum delay 具有极小延迟的比较器网络
Networks of workstations 工作站网络
Neumann, John von(= Margittai Neumann János) 诺伊曼，约翰·冯(= 玛奇特泰·诺依曼·贾诺斯)
Newcomb, Simon 纽科姆，西蒙
Newell, Allen 纽厄尔，阿伦
Newman, Donald Joseph 纽曼，唐纳德·约瑟夫
Nielsen, Jakob 尼尔森·雅各布
Nievergelt, Jürg 尼弗格尔特，朱尔格
Nijenhuis, Albert 尼詹休斯，阿尔伯特
Nikitin, Andrei lvanovich 尼基廷，安德烈·伊凡诺维奇
Nitty-gritty 事情的真相，本质
Nodine, Mark Howard 诺迪尼，马克·霍华德
Non-messing-up theorem 无混乱定理
Nondeterministic adversary 不确定的对手
Nörlund, Niels Erik 诺伦德·奈尔斯·埃立克
Normal deviate 标准离差
Normal distribution, approximately 近似的正态分布
Norwegian language 挪威语
Noshita, Kohei 野下浩平
Notations, index to 记号索引
Novelli, Jean-Christophe 诺维利，吉恩-克里斯托弗
NOW-Sort NOW－排序
NP-complete problems NP－完全问题
Nsort N 排序
Null permutation 空排列
Number-crunching computers 咬嚼数字的计算机
Numerical instability 数值的不稳定性
Nyberg, Christopher 奈伯格，克里斯托弗
Oberhettinger, Fritz 奥伯赫特廷格尔，弗里茨
Oblivious algorithms 漠视算法
O'Connor, Daniel J. 奥康纳尔，丹尼尔·J
Octrees 八叉树
Odd-even merge 奇偶合并
Odd-even transposition sort 奇伪转量排序
Odd permutations 奇排列
Odell, Margaret K 奥德尔，玛格丽特·K
Oderfeld, Jan 奥德尔菲尔德，简
Odlyzko, Andrew Michael 奥德里兹戈，安德烈·迈克尔
Oettinger, Anthony Gervin 马特廷格尔，安托尼·杰尔文
Oldham, Jeffrey David 奥尔德罕，杰弗里·戴维
Olivié, Hendrik Johan 奥里维，亨德里克·约翰
Olson, Charles A. 奥尔森，查尔斯·A
Omega network 欧米茄网络
One-sided height-balanced trees 一边高度平衡树
One-tape sorting 一条带排序
O'Neil, Patrick Eugene 奥奈尔，帕克里克·尤金
Ones' complement notation 一列补码的记号
Online merge sorting 联札合并排序
Open addressing 开式寻址
optimum 最优开式寻址
Operating systems 操作系统
Optimization of loops 循环的优化
Optimization of tests 测试的优化
Optimum binary search trees 最优二叉查找树

Optimum digital search trees　最优数字查找树
Optimum exchange sorting　最优交换排序
Optimum linear arrangements　最优线性安排
Optimum linear probing　最优线性探查
Optimum linked trie　最优链接检索结构
Optimum merge patterns　最优合并模式
Optimum open addressing　最优开式寻址
Optimum permutations　最优排列
Optimum polyphase merge　最优多阶段合并
Optimum searching　最优查找
Optimum sorting　最优排序
OR(bitwise or)　OR′(按二进位的或)运算
Order ideals　次序理想
Order relations　次序关系
Order statistics　次序统计
Ordered hashing　顺序的散列
Ordered partitions　顺序的分划
Ordered table, searching an　查找一个顺序表
Ordering of permutations　排列的顺序
Organ-pipe order　风琴管次序
Oriented trees　有向树
Orosz, Gábor　奥罗斯泽,加博尔
O'Rourke, Joseph　奥罗尔克·约瑟夫
Orthogonal range queries　正交范围查询
Oscillating radix sort　振荡基数排序
Oscillating sort　振荡排序
Overflow, arithmetic　算术溢出
　in *B*-trees　*B* 树中的溢出
　in hash tables　散列表中的溢出
Overmars, Markus(= Mark) Hendrik　奥弗尔马斯,马尔库斯(=马克)亨德里克
Own coding　特种编码

P-way merging　*P* 路合并
Packing　包装
Paging　分页
Pagodas　塔
Paige, Robert Allarr　一派格,罗伯特·阿拉尔
Painter, James Allan　佩因特,詹姆斯·阿伦
Pairing heaps　双堆
Pak, Igor Markovich　帕克,伊呆尔·马科维奇
Palermo, Frank Pantaleone　佩勒莫,弗朗克·潘塔里昂
Pallo, Jean Marcel　帕尔罗,吉恩·马基尔
Panny, Wolfgang Christian　三番尼,沃尔弗岗·克里斯蒂安
Papernov, Abram Alexandrovich　佩珀诺夫,阿布兰·阿历山德洛维奇
Pappus of Alexandria　阿历山德里亚的帕普斯
Parallel processing　并行处理
　merging　合并并行处理
　searching　查找并行处理
　sorting　排序并行处理
Parberry, Ian　帕贝利,伊恩
Pardo　帕多,见 Trabb Pardo
Pareto, Vilfredo　帕里多,维尔菲里多
Pareto distribution　帕里多分布
Parker, Ernest Tilden　帕克,欧内斯特·蒂尔登
Parkin, Thomas Randall　帕金,托马斯·兰道尔
Parking problem　停车问题
Parsimonious algorithms　吝啬的算法
Partial match retrieval　部分匹配检索
Partial ordering　偏序
　of permutations　排序的偏序
Partition-exchange sort　分划一交换排序
Partitioning a file　划分一个文件
　into three blocks　划分成三个块
Partitions of a set　一个集合的分划
Partitions of an integer　一个整数的分划
　ordered　次序的分划
　plane　平面的分划
Patashnik, Oren　帕塔什里克,奥伦
Patents　专利杂志
Paterson, Michael Stewart　帕特森,米歇尔·斯图尔特
Path length of a tree　一棵树的通路长度,见 External path length, Internal path length
　minimum　极小通路长度
　weighted　加权通路长度
　weighted by degrees　通过度来加权的通路长度
Patricia　一种特殊的树结构

Pólya, György(= George) 波利亚,乔治
Polygons, regular 正规的多边形
Polynomial arithmetic 多项式算术
Polynomial hashing 多项式散列
Polyphase merge sorting 多阶段合并排序
  Caron variation 卡伦形式的多阶段合并排序
  optimum 最优多阶段合并排序
  read-backward 向后读多阶段合并排序
  tape-splitting 多阶段合并排序带的分开
Polyphase radix sorting 多阶段基数排序
Pool, Jan Albertus van der 普尔,简·阿尔伯特斯·厄·德
Pool of memory 存储池
Poonen, Bjorn 普能,布约恩
Porter, Thomas K 波特,托马斯
Post office 邮局
Post-office trees 邮局树
Posting 置入,见 Insertion
Pouring liquid 倾注液体
Power of merge 合并的能力,见 Growth ratio
Powers, James 鲍尔斯,詹姆斯
Pratt, Richard Don 普拉特,理查德·顿
Pratt, Vaughan Ronald 普拉特,沃恩·罗纳德
  sorting method 普拉特排序方法
Prediction 预测,见 Forecasting
Preferential arrangements 优先安排
Prefetching 预取
Prefix 前缀
Prefix code 前缀码
  for all nonnegative integers 对于所有非负整数的预取码
Prefix search 前缀查找,见 Trie search
Preorder merge 前根顺序合并
Prestet, Jean 普雷斯提特,琼
Prime numbers 质数,素数
Primitive comparator networks 本原排序网络
Principle of optimality 最优性原理
Pring, Edward John 普林,爱德华·约翰
Prins, Jan Fokko 普林斯,简·福克戈
Priority deques 优先双队
Priority queues 优先队
  merging 合并优先队
Priority search trees 优先查找树
Probability density functions 概率密度函数
Probability distributions 概率分布
  beta β概率分布
  binomial 二项式概率分布
  fractal 分数概率分布
  normal 正态概率分布
  Pareto 概率分布
  Poisson 泊松概率分布
  random 随机概率分布
  uniform 一致概率分布
  Yule 尤利概率分布
  Zipf Zipf 概率分布
Probability generating functions 概率生成函数
Prodinger, Helmut 普罗定格尔,赫尔默特
Product of consecutive binomial coefficients 连续的二项式系数的乘积
Proof of algorithms 算法的证明
Prusker, Francis 普鲁斯克尔,弗朗西斯
Prywes, Noah Shmarya 普里威斯,诺亚·斯马里亚
Pseudolines 虚拟线
Psi function $\psi(z)$ 普赛函数 $\psi(z)$
Puech, Claude Henri Clair Marie Jules 普齐,克劳德·亨利·克莱尔·玛丽·朱利斯
Pugh, William Worthington, Jr. 小普格,威廉·沃特辛顿
Punched cards 穿孔卡片

q-multinomial coefficients q 多项式系数
q-nomial coefficients q 项式系数
q-series q 级数
Quadrangle inequality 四边形不等式
Quadratic probing 二次探查
Quadratic selection 二次选择
Quadruple systems 四元组系统
Quadtrees 四叉树
Queries 查询
Questionnaires 调查表

determinant 范德蒙德行列式
Variable-length code 可变长代码
Variable-length keys,searching for 用于查找的可变长键码
Variable-length records 可变长记录
Variable-length strings,sorting 变长字符串程序
Variance,different notions of 变量的不同记号
Vector representation of merge patterns 合并模式的向量表示
Velthuis,Frans Jozef 维尔瑟尤斯·弗朗斯·乔泽夫
Venn,John Leonard 维恩,维翰·伦纳德
Vershik,Anatoly Moiseevich 维尔锡克,安纳托里·莫伊西维奇
Viennot,Gérard Xavier 维诺特,杰加德·扎维尔
Viola Deambrosis,Alfredo 维奥拉·迪亚姆布罗西斯,阿尔弗雷多
Virtual memory 虚拟存储器
Vitter,Jeffrey Scott 维特尔,杰弗里·斯戈特(魏杰甫)
von Mises,Richard,Edler 冯·米塞斯,理查德,埃德勒
von Neumann,John(=Margittai Neumann János 冯·诺伊曼·约翰(=玛奇特泰·诺伊曼·贾诺斯)
VSAM VSAM(虚拟存储存取方法)
Vuillemin,Jean Etienne 武伊尔勒明,琼·埃递恩尼
Vyssotsky,Victor Alexander 维索特斯基,维克多·亚历山大
$W_t(n)$ and $\hat{W}_t(n)$ $W_t(n)$和 $\hat{W}_t(n)$函数
Wachs,Michelle Lynn 沃彻斯,迈彻利·林
Waks,David Jeffrey 威克斯,戴维·杰弗里
Waksman,Abraham 瓦克斯曼,亚布拉罕
Walker,Ewing Stockton 沃克,尤因·斯托克顿
Walker,Wayne Allan 沃克,韦因·艾伦
Wallis,John 沃里斯,约翰
Walters,John Rodney,Jr. 小沃尔特斯,约翰·罗德尼
Wang,Ya Wei 王亚威
Wang,Yihsiao 王义孝
Ward,Morgan 沃德,摩根
Watanabe,Masatoshi 渡边,雅俊
Waters,Samuel Joseph 沃特斯,塞缪尔·约瑟夫
Waugh,Evelyn,Arthur St.John 沃格,埃威林·阿瑟·圣约翰
Weak Bruhat order 维克·布鲁哈特顺序
Wedekind,Hartmut 韦得金德,哈特马特
Wegener,Ingo Werner 维吉纳尔,印戈·维尔纳
Wegman,Mark N 维格曼,马克·N
Wegner,Lutz Michael 维格纳,卢特兹·迈克尔
Weight-balanced trees 加权平衡树
Weighted path length 加权通路长度
Weisner,Louis 韦斯纳.路易斯
Weiss,Benjamin 韦斯,本杰明
Weiss Harold 韦斯·哈罗德
Weiss,Mark Allen 韦斯,马克·艾伦
Weissblum,Walter 韦斯布卢姆,沃尔特
Wells,Mark Brimhall 威尔士,马克·布里姆霍尔
Wessner,Russell Lee 韦斯纳,拉塞尔·李
Wheeler,David John 惠勒,戴维·约翰
Whirlwind computer 沃勒温德计算机
Whitlow,Duane L 韦特罗,瑞尼·L
Wiedemann,Douglas Henry 韦伊递曼,道格拉斯·亨利
Wiener,Norbert 韦伊纳,诺伯特
Wigram,George Vicesimus 韦格拉姆,乔治·维斯西姆斯
Wijngaarden,Adriaan van 维因加尔登,艾德里安·范
Wiles,Andrew John 维里斯,安德鲁·约翰
Wilf,Herbert Saul 维尔夫,希尔伯特·绍尔
Willard,Dan Edward 维尔拉德,丹·爱德华
Williams,Francis A.,Jr. 小威廉斯,弗朗西斯·A
Williamson,Stanley Gill 威廉逊,斯坦利·吉尔
Wilson,David Bruce 威尔逊,戴维·布鲁斯
Windley,Peter Francis 温得利,彼得·弗朗西斯
Winkler,Phyllis Astrid Benson 温克勒,菲利斯,阿斯特里德·本森
Wong,Chak-Kuen 黄泽权
Wood,Derick 伍德,德里克
Woodall,Arthur David 伍德尔,阿瑟·戴维
Woodrum,Luther Jay 伍德朗姆,卢瑟·杰伊
Wormald,Nicholas Charles 沃马德,尼古拉斯·查尔斯

Wrench, John William, Jr.　小伦奇，约翰·威廉
Wright Edward Maitland　赖特，爱德华·梅特兰
Wu Jigang　武继刚
Wyman, Max　怀曼，马克斯
Yao, Andrew Chi-Chih　姚期智
Yao, Frances Foong Chu　姚储枫
Yoash, Nahal Ben　约亚斯，纳哈尔·本
Pseudonym of Gideon Yuval　吉迪安·尤瓦尔的假名
Youden, William Wallace　尤登，威廉·华莱士
Young, Alfred　杨，阿尔弗雷德
　tableaux　杨氏图表
Young, Frank Hood　杨，法兰克·胡德
Yuba Toshitsugu　弓场敏嗣
Yuen, Pasteur Shih Teh　袁师德
Yule, George Udny　尤利，乔治·尤德尼
　distribution　尤利分布
Zalk, Martin Maurice　扎克，马丁·莫里斯
Zave, Derek Alan　扎夫，德里克·艾伦
Zeckendorf, Edouard　泽肯多夫，埃多瓦德
Zeilberger, Doron　泽尔伯格，多伦
Zero-one principle　0-1 原理
Zeta function $\zeta(z)$　ζ 函数
Zhang, Bin　张斌
Zhang, Linbo　张林波
Zhu, Hong　朱洪
Zigzag paths　弯曲道路，也见 Lattice paths
Zijlstra, Erik　吉尔斯特拉，埃里克
Zipf, George Kingsley　泽弗，乔治·金斯利
　distribution　泽弗分布
Ziviani, Nivio　泽雅亚尼，尼维奥
Zoberbier, Werner　佐贝尔比尔，维尔纳
Zodiac　佐迪亚克
Zolnowsky, John Edward　佐诺斯基·约翰·爱德华
Zuse, Konrad　朱斯·康拉德
Zweben, Stuart Harvey　泽维本·史图瓦特·哈维
Zwick, Uri　泽维克，尤里

Although you may pass for
an artist, computist, or analyst,
yet you may not be justly esteemed
a man of science.
尽管你作为一名艺术家、计算家或分析家可能是合格的，
但你未必能被公正地尊为一位科学家。
—GEORGE BERKELEY, *The Analyst* (1734)

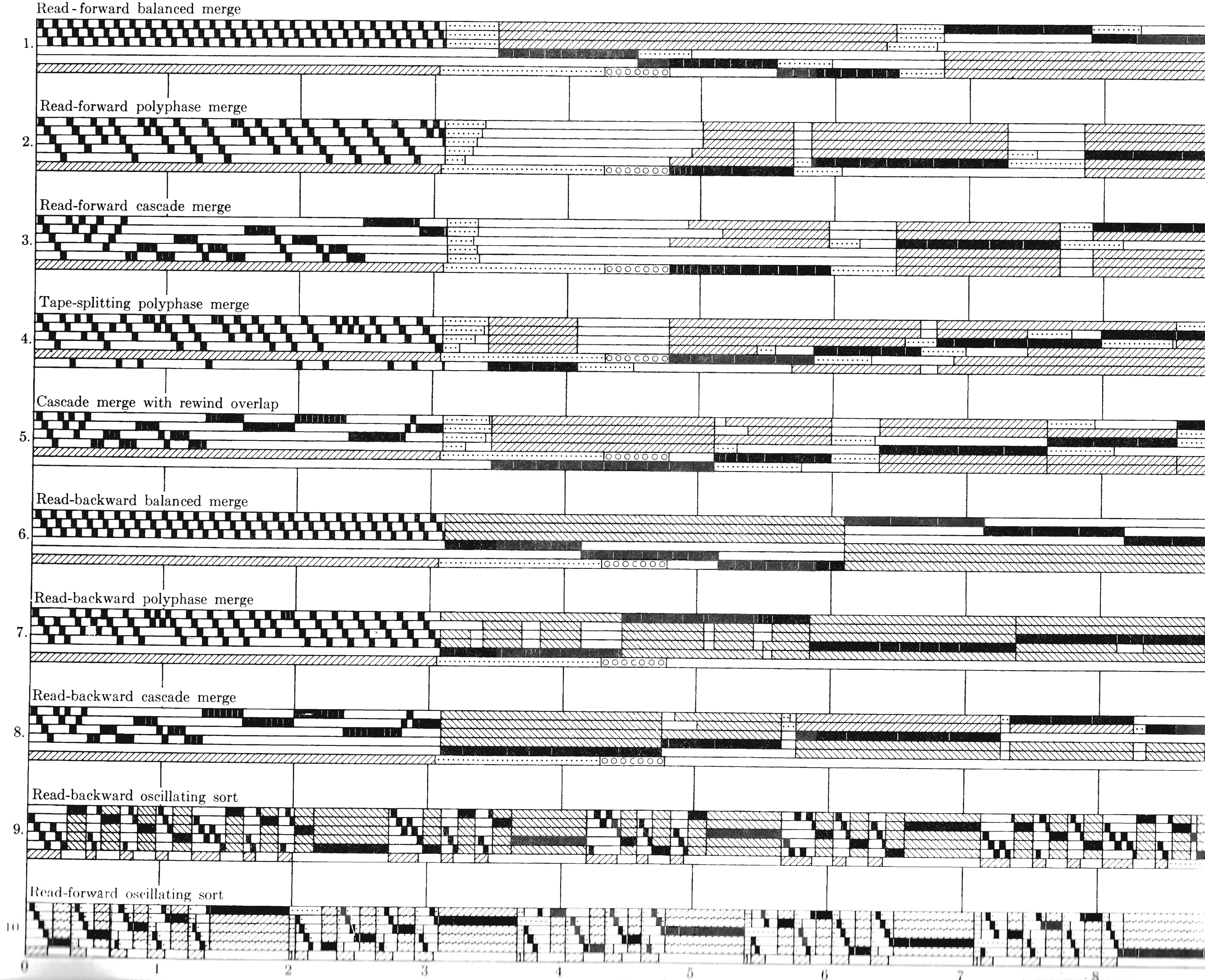
内 容 简 介

本书是国内外业界广泛关注的《计算机程序设计艺术》第 3 卷的最新版。本卷全面考察了经典计算机排序和查找技术。它扩充了第 1 卷对数据结构的处理，以将大小数据库和内外存储器一并考虑；遴选了经精心核验的计算机方法，并对其效率做了定量分析。本卷的突出特点是对“最优排序”一节的修订和对排列论与通用散列法的讨论。

本书附有大量习题和答案，标明了难易程度和数学概念的使用。

本书内容精辟，语言流畅，引人入胜，可供从事计算机科学及相关学科的工作人员参考、研究和借鉴，也是相关专业高等院校的理想教材和教学参考书。

本次印刷已根据 http://sunburn.stanford.edu/~knuth/taocp.html 上的最新勘误表（截止到 2003 年 1 月 25 日）校正了原版书中的错误。



Read-forward balanced merge
1.
Read-forward polyphase merge
2.
Read-forward cascade merge
3.
Tape-splitting polyphase merge
4.
Cascade merge with rewind overlap
5.
Read-backward balanced merge
6.
Read-backward polyphase merge
7.
Read-backward cascade merge
8.
Read-backward oscillating sort
9.
Read-forward oscillating sort
10
0
1
2
3
4
5
6
7
8

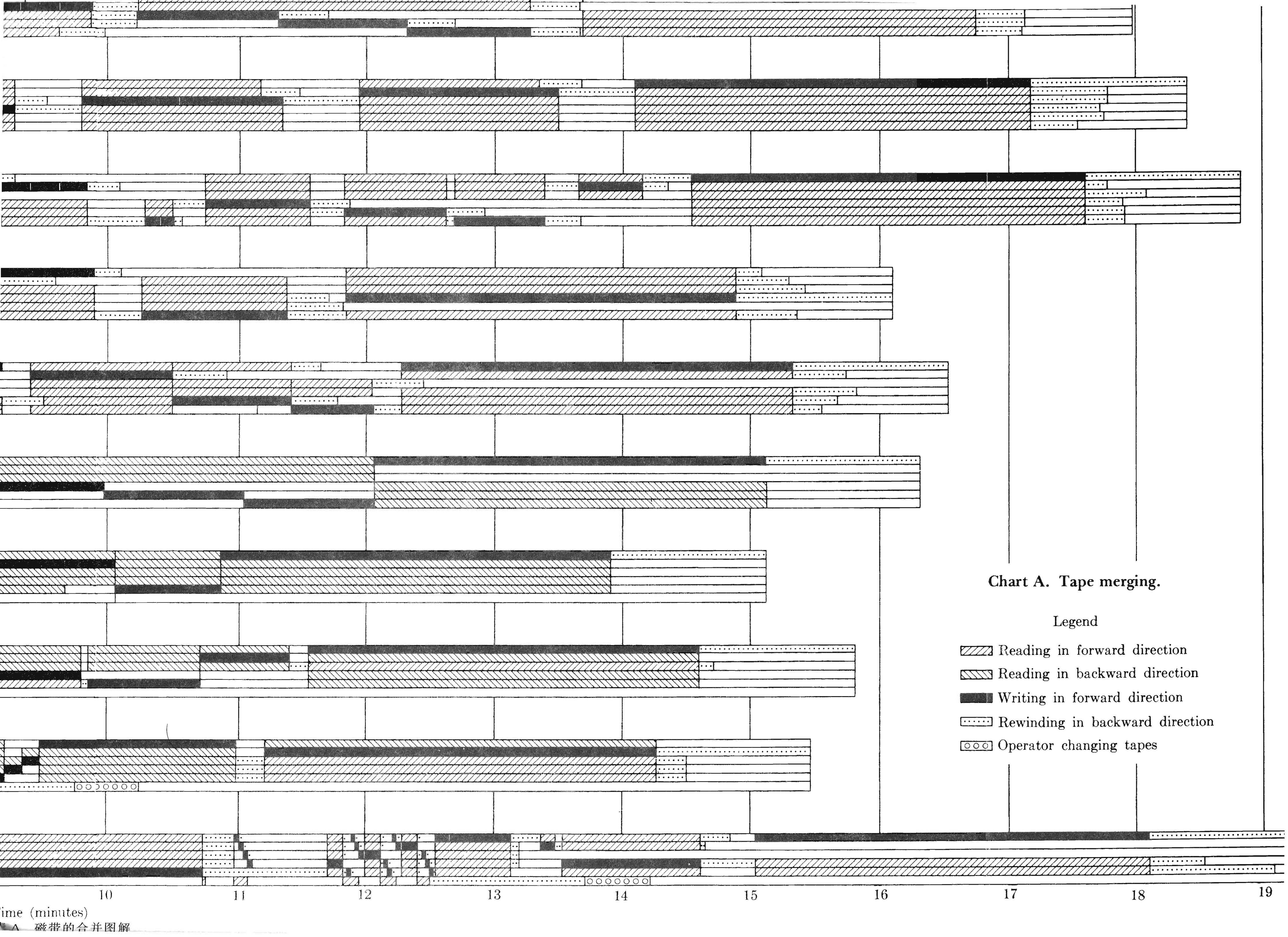


Chart A. Tape merging.

字符代码	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
	␣	A	B	C	D	E	F	G	H	I	Δ	J	K	L	M	N	O	P	Q	R	Σ	Π	S	T	U

00	*1*	**01**	*2*	**02**	*2*	**03**	*10*
无操作 NOP(0)		rA←rA + V ADD(0:5) FADD(6)		rA←rA − V SUB(0:5) FSUB(6)		rAX←rA × V MUL(0:5) FMUL(6)	
08	*2*	**09**	*2*	**10**	*2*	**11**	*2*
rA←V LDA(0:5)		rI1←V LD1(0:5)		rI2←V LD2(0:5)		rI3←V LD3(0:5)	
16	*2*	**17**	*2*	**18**	*2*	**19**	*2*
rA← − V LDAN(0:5)		rI1← − V LD1N(0:5)		rI2← − V LD2N(0:5)		rI3← − V LD3N(0:5)	
24	*2*	**25**	*2*	**26**	*2*	**27**	*2*
M(F)←rA STA(0:5)		M(F)←rI1 ST1(0:5)		M(F)←rI2 ST2(0:5)		M(F)←rI3 ST3(0:5)	
32	*2*	**33**	*2*	**34**	*1*	**35**	*1* + T
M(F)←rJ STJ(0:2)		M(F)←0 STZ(0:5)		设备 F 忙碌吗? JBUS(0)		控制,设备 F IOC(0)	
40	*1*	**41**	*1*	**42**	*1*	**43**	*1*
rA:0,转移 JA[+]		rI1:0,转移 J1[+]		rI2:0,转移 J2[+]		rI3:0,转移 J3[+]	
48	*1*	**49**	*1*	**50**	*1*	**51**	*1*
rA←[rA]? ± M INCA(0) DECA(1) ENTA(2) ENNA(3)		rI1←[rI1]? ± M INC1(0) DEC1(1) ENT1(2) ENN1(3)		rI2←[rI2]? ± M INC2(0) DEC2(1) ENT2(2) ENN2(3)		rI3←[rI3]? ± M INC3(0) DEC3(1) ENT3(2) ENN3(3)	
56	*2*	**57**	*2*	**58**	*2*	**59**	*2*
CI←rA(F):V CMPA(0:5) FCMP(6)		CI←rI1(F):V CMP1(0:5)		CI←rI2(F):V CMP2(0:5)		CI←rI3(F):V CMP3(0:5)	

一般形式

C	*t*
描述 OP(F)	

C = 操作码,指令的(5:5)字段

F = 操作码的变形,指令的(4:4)字段

M = 变址后的指令地址

V = M(F) = 单元 M 的 F 字段的内容

OP = 操作的符号名

(F) = 标准的 F 设定

t = 执行时间　*T* = 互锁时间

25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55
V	W	X	Y	Z	0	1	2	3	4	5	6	7	8	9	.	,	(	)	+	−	*	/	=	$	<	>	@	:	:	'

04	*12*	**05**	*10*	**06**	*2*	**07**	*1 + 2*F
rAX←rAX/V rX←余数 DIV(0:5) FDIV(6)		特殊的 NUM(0) CHAR(1) HLT(2)		移位 M 个字节 SLA(0) SRA(1) SLAX(2) SRAX(3) SLC(4) SRC(5)		从 M 移动 F 字 到 rI1 MOVE(1)	
12	*2*	**13**	*2*	**14**	*2*	**15**	*2*
rI4←V LD4(0:5)		rI5←V LD5(0:5)		rI6←V LD6(0:5)		rX←V LDX(0:5)	
20	*2*	**21**	*2*	**22**	*2*	**23**	*2*
rI4← − V LD4N(0:5)		rI5← − V LD5N(0:5)		rI6← − V LD6N(0:5)		rX← − V LDXN(0:5)	
28	*2*	**29**	*2*	**30**	*2*	**31**	*2*
M(F)←rI4 ST4(0:5)		M(F)←rI5 ST5(0:5)		M(F)←rI6 ST6(0:5)		M(F)←rX STX(0:5)	
36	*1 + T*	**37**	*1 + T*	**38**	*1*	**39**	*1*
输入,设备 F IN(0)		输出,设备 F OUT(0)		设备 F 就绪? JRED(0)		转移 JMP(0) JSJ(1) JOV(2) JNOV(3) 还有下边的[*]	
44	*1*	**45**	*1*	**46**	*1*	**47**	*1*
rI4:0,转移 J4[+]		rI5:0,转移 J5[+]		rI6:0,转移 J6[+]		rX:0,转移 JX[+]	
52	*1*	**53**	*1*	**54**	*1*	**55**	*1*
rI4←[rI4]? ± M INC4(0) DEC4(1) ENT4(2) ENN4(3)		rI5←[rI5]? ± M INC5(0) DEC5(1) ENT5(2) ENN5(3)		rI6←[rI6]? ± M INC6(0) DEC6(1) ENT6(2) ENN6(3)		rX←[rX]? ± M INCX(0) DECX(1) ENTX(2) ENNX(3)	
60	*2*	**61**	*2*	**62**	*2*	**63**	*2*
CI←rI4(F):V CMP4(0:5)		CI←rI5(F):V CMP5(0:5)		CI←rI6(F):V CMP6(0:5)		CI←rX(F):V CMPX(0:5)	

	[*] [+]:
rA = 寄存器 A	JL(4) < N(0)
rX = 寄存器 X	JE(5) = Z(1)
rAX = 寄存器 A 和 X 当做一个	JG(6) > P(2)
rIi = 变址寄存器 i,1≤i≤6	JGE(7) ≥ NN(3)
rJ = 寄存器 J	JNE(8) ≠ NZ(4)
CI = 比较指示器	JLE(9) ≤ NP(5)

第 1 卷 / 基本算法（第 3 版）

丛书第 1 卷以基本的程序设计概念和技术开始，然后专注于信息结构——计算机内部信息的表示、数据元素之间的结构关系及其有效处理方法。描述了模拟、数值方法、符号计算、软件与系统设计的初等应用。新版本增加了几十项简单但重要的算法和技术，并对有关数学预备知识做了大量修正以适应现时研究的趋向。

第 2 卷 / 半数值算法（第 3 版）

第 2 卷对半数值算法领域做了全面介绍，分“随机数”和“算术”两章。本卷总结了主要算法范例及这些算法的基本理论，广泛剖析了计算机程序设计与数值分析间的相互联系。第 3 版中特别值得注意的是 Knuth 对随机数生成程序的重新处理和对形式幂级数计算的讨论。

第 3 卷 / 排序和查找（第 2 版）

第 3 卷的头一次修订对经典计算机排序和查找技术做了最全面的考察。它扩充了第 1 卷对数据结构的处理，以将大小数据库和内外存储器一并考虑；遴选了精心核验的计算机方法，并对其效率做了定量分析。第 3 卷的突出特点是对“最优排序”一节的修订和对排列论与通用散列法的讨论。

经典计算机
科学著作
最新修订版

计算机科学/程序设计

它本来是作为参考书撰写的，但有人却发现每一卷都可以饶有兴致地从头读到尾。一位中国的程序员甚至把他的阅读经历比做读诗。

如果你认为你确实是一个好的程序员，读一读Knuth的《计算机程序设计艺术》吧，要是你真把它读通了，你就可以给我递简历了。 ——Bill Gates

这一鸿篇巨制被广泛誉为对经典计算机科学的权威描述，头三卷在几十年来一直是学生、研究人员和业内人士的无价财富。

一部所有基础算法的宝典，今天的许多软件开发者关于计算机程序设计的绝大多数知识都是从该宝典中获得的。 ——*Byte*, September 1995

不计其数的读者深受Knuth著作的影响。科学家们惊讶于他的分析的精美与雅致，而普通程序员每天都从其提供的“菜谱”中获得问题解决方案。书的恢宏、透彻、精确与幽默赢得了所有人的尊敬。

我无法描述它们在我学习和娱乐中伴我度过的愉悦时光。我在车里，在餐馆里，在家里……甚至在我儿子棒球小联赛的间隙都忘不了带上它们，一有空就捧出来阅读。 ——Charles Long

不管你基础如何，倘若你想认真地编写任何计算机程序，你都有理由把这套书的任何一卷抱回家，以便在你学习和工作的时候随时翻阅。

要是有一个问题难到要把Knuth的著作请下书架，那就太令人愉悦了。我发现，人们只要翻一翻它，就会对计算机产生极其有用的令人“恐慌”的影响。 ——Jonathan Laventhol

为反映该领域的最新发展，Knuth二十多年来第一次将三卷书全部做了修订。他的修订主要集中在上一版以来趋于一致的知识，已经解决的问题，以及有所变化的问题。为保持本书的权威性，关于该领域先驱工作的历史信息都做了更新。为维护作者苦心孤诣追求至善至美的盛誉，新的版本将敏锐和苛刻的读者发现的少量技术错误都做了更正。增加了上百的新习题，对您也是新的挑战。

责任编辑：辛再甫 zfxin@ndip. com.cn

计算机程序设计艺术

第3卷 排序与查找

（第2版）

ISBN 7-118-02812-6

9 787118 028126 >

ISBN 7-118-02812-6/TP·713
定价:98.00 元

国防工业出版社
National Defence Industry Press
http://www.ndip.com.cn

Addison-Wesley

Addison Wesley